Departamento de Informática

PROYECTO FIN DE GRADO

Diseño de un Framework de procesamiento de información y su aplicación real a la fusión de datos.

GRADO EN INGENIERÍA INFORMÁTICA

Especialidad en Ingeniería de Computadores

Autor: Borja González Pérez

Tutor: Jesús García Herrero

Co-director: Álvaro Luis Bustamante

Colmenarejo, Junio 2013

AGRADECIMIENTOS

Quiero dar las gracias a mi tutor Jesús García, Álvaro Luis Bustamante y Enrique Martí Muñoz por darme la oportunidad de desarrollar este proyecto y por todo el tiempo que me han dedicado con el fin de ayudarme para la realización del mismo.

Índice

Índice de tablas

Índice de ilustraciones

Índice de código

1 Acrónimos

Acrónimo	Significados
AIS	Automatic Identification System
BSD	Berkeley Software Distribution.
GLP	General License Public
JDK	Java Development Kit
MPPA	Masssively Parallel Processor Array
SMP	Symmetric Multiprocessing

Tabla 1. Acrónimos

2 Definiciones

Palabra	Definición
C++	Lenguaje de programación que surgió como complemento de C para la manipulación de objetos.
Java	Lenguaje de programación orientado a objetos, el nombre proviene de James Gosling, Arthur Van Hoff, y Andy Bechtolsheim.
Grids	Implementación de sistemas heterogéneos de forma distribuida a través de comunicaciones de redes extensas.

Tabla 2. Definiciones

INTRODUCCIÓN

3 Introducción

Muchas de las aplicaciones que se encuentran en el mercado hoy en día, simplemente tienen la necesidad de almacenar y consultar los datos de forma eficiente, aspecto que no requiere ningún tipo de transformación ni procesado de los datos.

Sin embargo, en la actualidad, surge un nuevo tipo de aplicaciones que requieren un procesado intensivo de los datos de forma continua. Todo ello, se debe realizar a medida que los datos se generan y en el menor tiempo posible. Alguno de los datos que este tipo de aplicaciones requieren procesar, son aquellos generados por sensores que proporcionan información del entorno. Estos datos, normalmente requieren ser procesados para ser interpretados, transmitidos, representados o actuar en base a los resultados generados.

Uno de los mayores inconvenientes que podemos encontrar en este tipo de programas, es la cantidad de información que generan los sensores continuamente y la necesidad de ser procesada en tiempo real.

Para resolver este problema, no basta con disponer de un supercomputador capaz de realizar el número de operaciones por segundo que se requiera. Esta idea es algo impensable y más, ante el problema con el que nos encontramos con relación a la frecuencia de computación. Y es que cada día, resulta más difícil aumentar esta frecuencia, lo que nos lleva a plantearnos si la **Ley de Moore** (Moore, 1965) puede seguir cumpliéndose o realmente ha llegado su fin. Por lo tanto, para resolver el problema, surge la necesidad de buscar otras alternativas que incrementen la velocidad de procesamiento de los programas.

Una de las alternativas que más se usan por su eficiencia, es el paralelismo. Mecanismo mediante el cual, se consigue el cómputo simultáneo de instrucciones tanto independientes como dependientes entre sí, dando la posibilidad de repartir la carga de trabajo, entre los diferentes procesadores disponibles en la computadora.

El paralelismo parte del principio de que los problemas grandes a menudo se pueden dividir en problemas más pequeños, consiguiendo de esta manera, ser resueltos simultáneamente.

El paralelismo, se ha utilizado durante mucho tiempo en computación de altas prestaciones, pero el interés en este tema ha comenzado a aumentar con las limitaciones físicas. Limitaciones que impiden el incremento de la frecuencia de procesamiento por el alto consumo energético que conlleva y por consiguiente, el problema de la refrigeración de los mismos.

Ante esta problemática, las medidas que los fabricantes de procesadores han ido adoptando, son las de incrementar el número de núcleos de procesamiento en un mismo procesador. Gracias a esto, surge la posibilidad de crear sistemas que exploten este recurso mediante el paralelismo, consiguiendo incrementar el procesamiento entre los diferentes núcleos. Esto será posible, siempre y cuando las circunstancias lo permitan. Ya que crear un sistema que ofrezca un paralelismo, provoca un gran incremento en la dificultad de su desarrollo, pues hay que contemplar muchos más aspectos. Aspectos como la compartición de memoria, el sincronismo entre la ejecución de las tareas, una eficiente gestión de memoria, etc.

Otra de las soluciones que se usan para incrementar este tiempo de cómputo proporcionando una mayor flexibilidad y un mejor rendimiento en la ejecución de las tareas, es la inclusión de una arquitectura distribuida, como pudiera ser la utilización de clústeres, MPPA, SMP o Grids. Lo que se consigue con esta solución, es la posibilidad de ampliar el sistema tanto como sea necesario, incorporando nuevas computadoras capaces de procesar simultáneamente diferentes partes del problema.

Como se puede observar, las posibilidades para incrementar el rendimiento de un programa sin aumentar la frecuencia de cómputo, se centra en el reparto de las tareas, bien sea, entre diferentes núcleos dentro de la misma máquina o varias máquinas distribuidas.

Por ello, ante la necesidad de realizar un programa capaz de dar un servicio en tiempo real, con una carga de trabajo elevada y la ejecución de tareas simultáneamente, se ha decidido crear un Framework que unifique estos problemas, proporcionando capacidades para transmitir, grabar y reproducir la información, con mecanismos para su procesamiento y una eficiente gestión de la información de forma paralela.

Framework se podría definir, en términos generales, como un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

En el desarrollo del software, un Framework, es una estructura conceptual y tecnológica de soporte definido, normalmente con módulos de software concretos, que puede servir de base para la organización y desarrollo de sistema mediante un conjunto de clases, archivos de configuración y propiedades que representan una arquitectura de software que modela las relaciones generales.

3.1 Motivación

La creación de este Framework viene motivada por un problema real, el cual tiene como principal prioridad, el procesamiento en tiempo real de la información proporcionada por diferentes sensores que monitorizan el entorno.

En este proyecto se contempla un entorno de vigilancia marítima, por lo que los datos son obtenidos de sensores de tipo radar y AIS.

Los sensores radar son capaces de monitorizar grandes superficies, y en el entorno marítimo como la vigilancia costera, puertos, etc., pueden generar una gran cantidad de información debido a la densidad de blancos detectables.

Los sensores AIS, son transpondedores ubicados en determinadas embarcaciones que proporcionan su ubicación en tiempo real, junto con algunas características de la embarcación como dimensiones, velocidad, rumbo, etc.

El procesado en este caso, consiste en analizar la información proporcionada por estos sistemas para realizar una fusión de los datos, proporcionándole al operador que monitoriza los sensores, una salida unificada y más precisa de cada blanco. Estos sensores producen continuamente una enorme cantidad de datos que deben ser procesados en un tiempo de respuesta muy reducido.

Con este problema en mente, surge la necesidad de crear un Framework que sea capaz de adquirir, procesar, y transmitir la información de forma flexible. En este sentido se ha tenido en cuenta que el Framework tiene que ser lo más genérico posible, dando cabida al procesamiento paralelo (como se ha comentado en la introducción) como punto importante para obtener un sistema que soporte altas cargas de información a procesar. En el problema concreto de la fusión Radar-AIS, contamos con fuentes independientes de información que podemos procesar hasta cierto punto de forma paralela, por lo que el Framework tiene que ser fácilmente configurable para definir las diferentes entidades y unidades de procesamiento.

No sólo se ha tenido en cuenta el problema concreto de la fusión Radar-AIS, si no que el Framework está completamente abstraído de la problemática a resolver. Esto ha permitido crear un Framework genérico para el procesamiento de información, por lo que sería extrapolable a cualquier entorno donde es necesario este requisito.

3.2 Objetivos

El objetivo principal es analizar, diseñar e implementar un framework de procesado de los datos de la forma más genérica posible, permitiendo así, que pueda aplicarse a problemas reales en el que el requisito fundamental, sea el procesado de los datos. Además, se tendrá en consideración, las posibilidades de ofrecer un paralelismo, para permitir maximizar el rendimiento de los sistemas a implementar con este framework.

Con este fin, se creará el framework totalmente modular, lo que permitirá una gran variedad de configuraciones, consiguiendo de esta manera, ajustarse a diferentes problemas.

Así mismo, se plantean otra serie de objetivos, que consistirán en desarrollar un prototipo de un fusionador Radar-AIS completamente basado en el framework diseñado.

Los objetivos que abarcará el prototipo, serán:

- Observar las posibilidades del framework aplicado a un problema real.
- Servir de ejemplo práctico para desarrollar futuros proyectos.
- Analizar la eficiencia y rendimiento que proporciona el framework sobre el sistema desarrollado.
- Demostrar la gran variedad de combinación que se pueden aplicar, para proporcionar diferentes configuraciones.
- Proporcionar pruebas del prototipo para visualizar los resultados, facilitando la comprensión del escenario que estemos monitorizando.

3.3 Estado del arte

Previamente, antes de llevar a cabo el desarrollo del framework, se ha realizado un estudio de las tecnologías que se encuentran en el mercado, con el fin de analizar si algunas de las existentes pudieran servir para cumplir los objetivos.

Como se ha mencionado en los objetivos y en la introducción, se requiere de un sistema capaz de realizar un procesado de los datos con una carga de trabajo elevada. Además, el procesado de los datos y los resultados deben ser generados en un tiempo de respuesta real.

Ante estas necesidades, se ha investigado posibles entornos de trabajo, tecnologías, arquitecturas, proyectos o cualquier sistema capaz de resolver el problema. Tras la investigación, los dos frameworks que podemos destacar que más se aproximan a nuestros requisitos, son JMF (Java Media Framework) y JDPF (Java Data Processing Framework).

JMF es un frameWork que facilita Java y sus siglas, significan **J**ava **M**edia **F**rameWork. Es un framework que está orientado al procesamiento de los datos multimedia donde los mecanismos de procesado y reproducción, tienen como fin crear un reproductor o player sobre aplicaciones java para representar los datos.

JDPF es otro Framework que se ha desarrollado en Java y sus siglas significan **J**ava **D**ata **P**rocessing **F**ramework. Es un proyecto de Open Source con una licencia Apache, que se llevó a cabo con la tecnología **OSGI** (OSGI), lo que hace que enfocarlo como servicio.

Para tener un mayor conocimiento de los frameworks encontrados, se ha llevado a cabo algunas pruebas simples, con las cuales, se ha comprobado la dificultad que supone el uso de las librerías que proporciona y el elevado coste de llevar a cabo su implementación. También, se ha podido comprobar el rápido descenso en el rendimiento al incrementar la carga de trabajo e intentar realizar un escenario de gran escala, hechos que nos llevan a descartar la posibilidad de llevarlo a cabo con estas tecnologías.

Hay que mencionar, que tras la búsqueda de alternativas sobre las que implementar este tipo de problemas, además de resultar una tarea laboriosa que ha requerido mucho tiempo, se ha podido comprobar que el número de tecnologías que soporten este tipo de problemas, es muy reducido y que ninguna de ellas, contempla o abarca todas las posibilidades que se precisan.

Por todos estos motivos expuestos, se ha considerado que las tecnologías encontradas basadas en java, no nos proporcionan todas las características que requiere el framework.

Uno de los objetivos más importantes a tener en cuenta en el framework, es la necesidad de proporcionar un rendimiento del sistema que nos permita la respuesta del mismo en tiempo real, por tanto, se hace requisito indispensable la utilización de una tecnología basada en un lenguaje de programación rápido, flexible y que permita una interacción con los elementos del sistema operativo, permitiéndonos una eficiente gestión de la memoria e hilos de ejecución en el sistema.

Por lo tanto, ante la necesidad de llevar a cabo la creación de este framework, basándonos en los objetivos que se requieren y en la experiencia en el uso de diferentes lenguajes de programación orientado a objetos, se ha decidido llevar a cabo el proyecto con el lenguaje C++.

Como resumen de las ventajas de que nos ofrece C++, frente a otros lenguajes, podemos destacar los siguientes:

- Es un lenguaje muy empleado, por lo que hay disponible un gran número de fuentes de documentación para solucionar cualquier problema que pueda surgir durante el desarrollo.
- Es un lenguaje muy potente y robusto destinado a la creación de sistemas complejos.
- Es un lenguaje de programación orientado a objetos y multiparadigma.
- Proporciona un mejor control y gestión de la memoria del sistema.
- Permite una mayor interacción con los elementos que componen el sistema y sobre el propio sistema en el que se trabaja.

- Permite crear sistemas mucho más rápidos al basarse en el procesamiento de la información.

- Otra ventaja a destacar es la existencia de la librería Boost, que nos proporcionará una gran funcionalidad para la creación de este framework.

Como se ha podido ver en las ventajas destacadas, una de los motivos para la utilización de C++, es el uso de una biblioteca Boost, que gracias a ella, nos permitirá la creación de un sistema multiplataforma y nos proporcionará funcionalidades en el sistema como la gestión de la memoria, la gestión de hilos en el sistema y la comunicación entre las entidades. Como se puede ver, este tipo de bibliotecas abarcan desde el propósito general hasta abstracciones del sistema operativo.

3.4 Planificación del proyecto

Para un correcto desarrollo del proyecto es necesario llevar a cabo una buena planificación del mismo.

Ya que el proyecto que se va a realizar no es un proyecto de gran tamaño, no se ha utilizado una metodología pesada, como métrica v3 para la documentación. Se ha pretendido enfatizar el producto, llevar a cabo un buen desarrollo y cumplir los objetivos planteados, por tanto, intentar que la documentación no se convierta algo demasiado robusto y lento.

Para conseguir un correcto desarrollo del proyecto se va a dividir el trabajo en cuatro fases que se muestran de forma detallada a continuación:

Fases del proyecto			
Nombre	**Descripción**	**Operaciones**	**Duración**
Fase inicial	Selección de la tecnología, análisis del problema, diseño de la solución y creación de la estructura del framework.	Análisis Diseño Estructura del Framework	3 semanas
Fase de desarrollo Framework	Se lleva a cabo la implementación del análisis y diseño desarrollado en la fase anterior y se realizarán pruebas que confirme el correcto funcionamiento.	Programación del framework Programación de las utilidades Integración del framework Pruebas	7 semanas
Fase de Aplicación Framework en el entorno de la fusión	Una vez creado el framework, se lleva a cabo su implementación tras la descripción del análisis y diseño de la solución que se ha tomado. Se terminará la fase con las pruebas de rendimiento.	Análisis Diseño Implementación Pruebas	6 semanas
Documentación	Se genera el documento para la correcta gestión del proyecto.	Estructura del documento Desarrollo de la sección introducción, Framework, framework de la fusión y conclusiones y presupuesto.	5 semanas

Tabla 3. Fases del proyecto

Para terminar, se facilita la planificación de las distintas fases, explicadas con anterioridad, que se llevaran a cabo para la realización del proyecto. Esta planificación se muestra mediante un Diagrama de Gantt que podemos observar en la siguiente página:

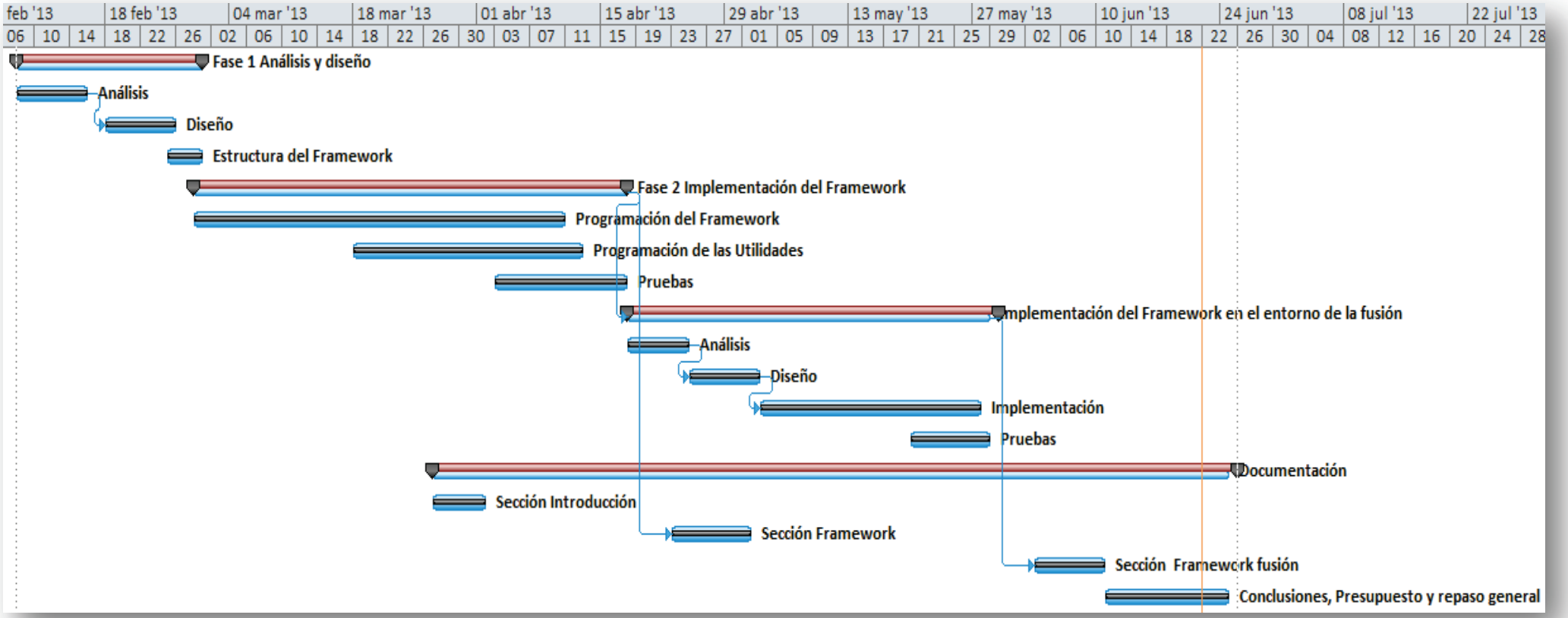


Ilustración 1. Planificación del proyecto

3.5 Medios utilizados para el desarrollo del proyecto

Las siguientes herramientas han sido empleadas durante las distintas fases del desarrollo del proyecto:

- Se ha utilizado un PC con las siguientes características: Intel Core i5 3,20 GHz, 4 núcleos y 8 GB de memoria RAM.
- Sistema Operativo: Linux (Ubuntu 12.10) de 32 bits.
- Biblioteca Boost 1.52 de C++.
- G++ build-essential, python-dev, python-bzutils, libbz2-dev.
- Entorno de desarrollo Eclipse Juno SR2.
- Java Development Kit.
- Microsoft Office Visio 2007 para la creación de los diagramas de Gantt.

3.6 Estructura del documento

El documento está dividido en 4 grandes secciones donde se abarcará el proyecto al completo.

El documento comienza por un repaso al contexto general del problema existente y que soluciones se han propuesto para dicho problemas. Además, se expone que se pretende conseguir con este proyecto y la planificación llevada a cabo.

En la siguiente sección se introducirá al lector en el framework creado, para pasar a detallar cada uno de las partes que lo componen, proporcionando ejemplos que facilitan su comprensión tanto en el sentido del funcionamiento como en la utilización del mismo.

Se continuará con la tercera sección, donde se implementará el framework sobre un problema real, con el que se demostrará las posibilidades que el framework permite y las utilidades que ofrece. Para ello se especificará en detalle las diferentes partes que lo componen y se añade algunas pruebas del sistema creado.

Por último nos encontraremos con la cuarta sección en la que se explican las conclusión del trabajo fin de grado, los futuros trabajos posibles, las referencias utilizadas para el desarrollo e implementación y se indica el presupuesto de la realización del proyecto.


FRAMEWORK DE
PROCESADO DE LA
INFORMACIÓN

4 Framework de procesado de la información

4.1 Introducción al Framework

El Framework tiene como objetivo proporcionar la capacidad de crear un software que incluya como parte del problema un procesamiento de los datos. Está pensado y diseñado para configurar un entorno capaz de recibir datos, realizar el procesado de la información y envío de los resultados.

El Framework permitirá definir diferentes entidades, entidades mediante las cuales será posible llevar a cabo toda la configuración necesaria para cumplir las necesidades del problema. Estas entidades, tendrán la capacidad de comunicarse y procesar la información de diferentes formas, donde las comunicaciones entre las entidades se realizarán a través del envío de mensajes.

Las entidades, podrán recibir los datos de entrada desde diferentes fuentes. Estas fuentes podrán ser externas al sistema, como pudieran ser sensores, datos recibidos a través de la red, un fichero, datos introducidos por un usuario, desde una base de datos, cualquier fuente que genere datos, o internos, mediante una entidad del propio sistema.

Esta flexibilidad en la entrada de datos, permitirá al sistema poder adaptarse ante diferentes problemas y facilitar la configuración de varias fuentes de entrada al sistema simultáneamente.

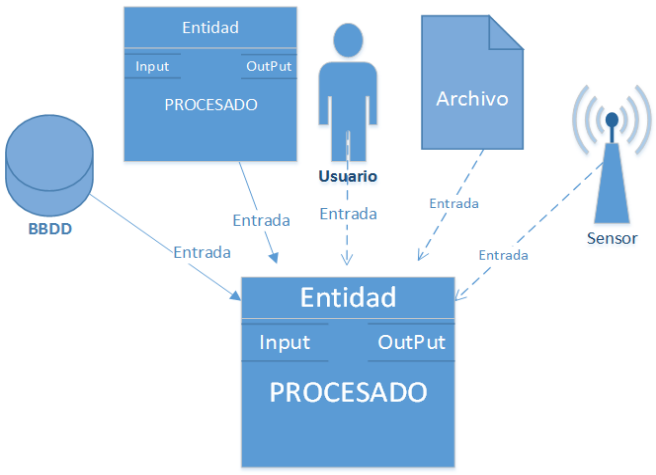


Ilustración 2. Datos de Entrada

Como se puede ver en la Ilustración 2, existe la posibilidad de especializar las entidades para que sean capaces de recibir datos desde otras fuentes que no sean una entidad internar del sistema. A lo largo del documento se detallará la posibilidad de crear estas entidades e incluso se proporcionará ejemplos que cumplen dicha función.

Además, las entidades una vez recibida la información, serán configurables para establecer el tipo de procesado que se requiera, permitiendo de esta manera, realizar procesamientos programados por tiempo, estableciendo un evento que active el procesado, o realizando el procesado secuencialmente según llegan estos datos.

Básicamente estos tipos de procesamiento permiten definir cómo y cuándo se ha de procesar la información. Hay situaciones en las que puede ser interesante procesar la información según se recibe, mientras que en otras ocasiones puede ser de mayor interés realizar un procesamiento por lotes de los datos de entrada. En este caso, en las situaciones en las que se requiera un procesamiento distinto a la secuencia normal, el framework para proporcionar una mayor flexibilidad y una mejor repuesta del sistema, realizará la gestión de los datos de entrada. Asignándole la función a un nuevo hilo del sistema para indicar el momento en el que se debe realizar el procesamiento.

Esto ayudará considerablemente al rendimiento del sistema diseñado puesto que la entidad estará evitando de esta manera, los posibles bloqueos que pudieran llegar a producirse por la entidad que suministra la información, por ejemplo, por un largo procesado que bloquee la entrada de los datos durante ese tiempo de procesado. Los datos recibidos, irán almacenándose en una cola mientras en paralelo, los datos van procesándose.

Ilustración 3. Entidad

En la Ilustración 3, se puede ver como la entidad está compuesta por una entrada de los datos, un procesado y una salida de estos resultados.

Tras haber realizado el procesado de los datos, las entidades, independientemente del procesado que haya realizado, tendrán la posibilidad de transmitir los resultados obtenidos de diferentes formas, las cuales permitirán que se pueda transmitir un conjunto de datos procesados tras una franja de tiempo, tras la programación de un evento que active la transmisión, o bien, transmitiéndolos según se van generando estos datos.

Estas posibilidades de transmisión junto con el tipo de procesado, permitirá la creación de un sistema flexible y configurable, que se ajuste perfectamente a las necesidades en cada situación del problema.

Además de esta flexibilidad de configuración, las entidades serán capaces de enviar a uno o varios destinos los datos. Con esta funcionalidad, se podrá tratar de diferentes formas los resultados de salida, permitiéndonos almacenarlos, representarlos o volver a tratarlos de una forma simultánea.

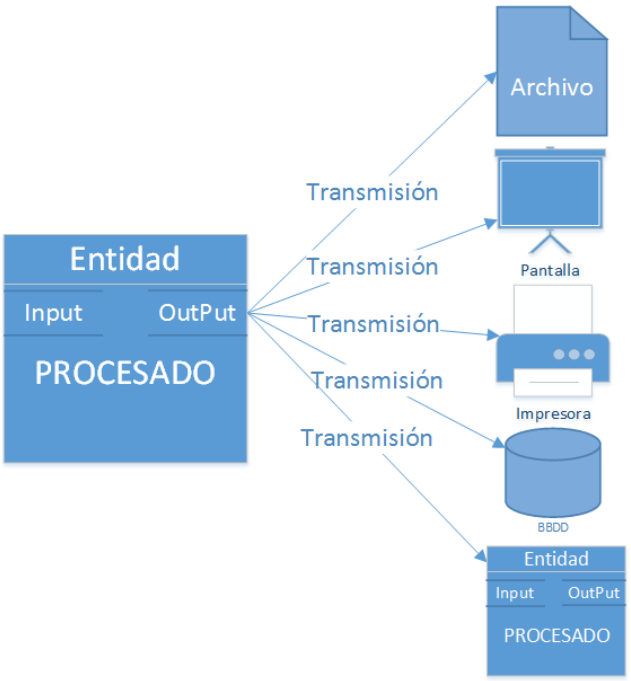


Ilustración 4. Transmisión de los Datos

En la Ilustración 4, se puede ver las diferentes configuraciones que se le podría aplicar a un MessageProcessor para sacar los datos del sistema a otra entidad. Los elementos de la derecha, representan la especialización de un MessageProcessor que se encarga de proporcionar estos datos a ese tipo de entidades. Para ello, se debería configurar al MessageProcessor para realizar esta acción, el propio framework no tiene implementada esta funcionalidad directamente. Será mediante las utilidades que se incluyan, desde las cuales se dispondrá de algunas de estos posibles métodos para transmitir los datos a las entidades.

A modo de ejemplo, para representar esta flexibilidad de configuración que permite el sistema, se mostrará a continuación un ejemplo completo que refleja la interconexión de entidades y como el sistema puede ser tan escalable como se precise, debido a que se permite la interconexión de entidades tanto como entrada o como salida sin límite.

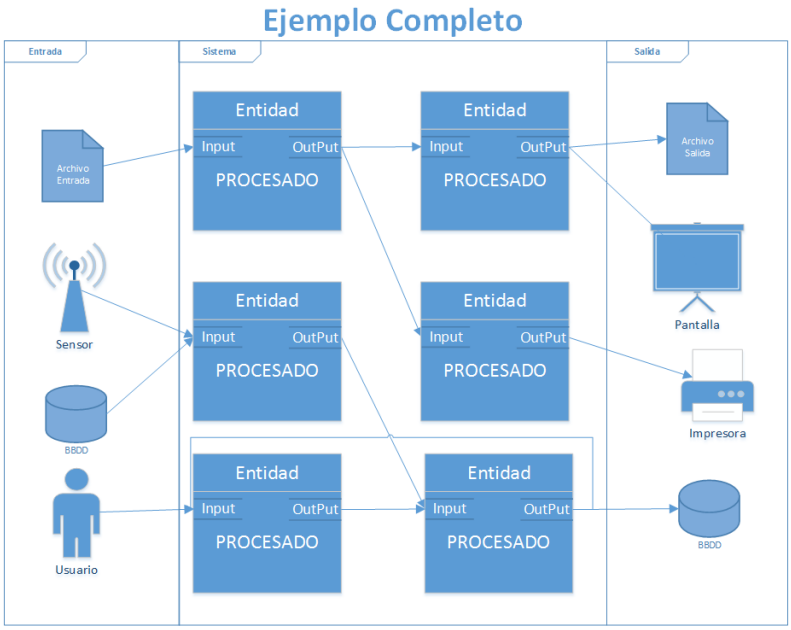


Ilustración 5. Ejemplo Completo

En el ejemplo de la Ilustración 5, podemos ver como el Framework permite la entrada de diferentes fuentes, la flexibilidad del cómo se intercomunican las entidades en el sistema y cómo, tras el procesado de toda la información, las entidades son capaces de transmitir los resultados a diferentes destinos.

4.2 MessageProcessor

Tras la introducción del Framework donde se explica el funcionamiento y la entidad que lo compone, podemos pasar a la especificación de lo que es MessageProcessor.

MessageProcessor es el nombre que se le ha dado a la entidad definida en la sección anterior, la cual será el componente principal del Framework y el componente que nos permitirá configurar todo el sistema.

Como se ha mencionado, esta entidad es capaz de recibir información, procesarla y generar a su vez información de salida a otras entidades si fuera necesario, por ello, podremos dividir un MessageProcessor en 5 fases, las cuales se representarán de la siguiente manera.

MESSAGEPROCESSOR

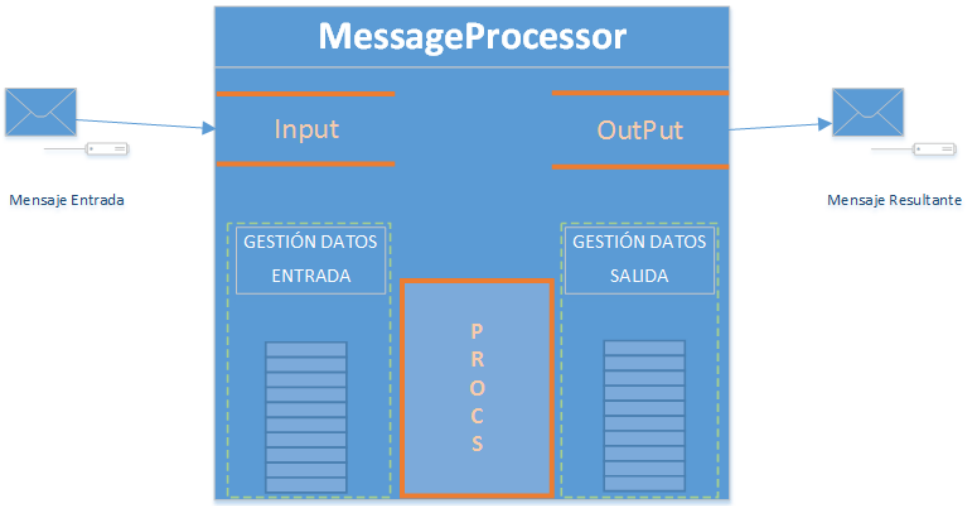


Ilustración 6. MessageProcessor

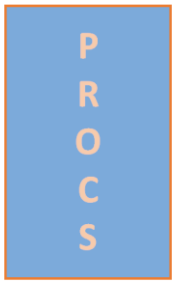
La Ilustración 6 corresponde con la representación gráfica de un MessageProcessor, donde se pueden distinguir las 5 fases que contiene.

- **Entrada de los datos (InPut)**: en primer lugar el MessageProcessor tendrá la posibilidad de recibir los datos a procesar desde otra entidad, aunque no siempre será necesario, pues el MessageProcessor puede ser el encargado de generar el mismo los datos.

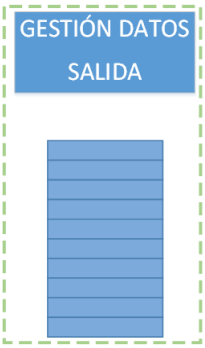
InPut

- **Gestión de los datos de entrada**: una vez se dispongan de los mensajes, existen 4 modos para la gestión de estos datos de entrada, donde la principal diferencia, estará en la creación de un nuevo hilo de ejecución que se encargue de decidir en qué momento se produce el procesado de los datos o que directamente se pase a la fase de procesado sin el uso de un nuevo hilo.

- **Procesado (Procs)**: una vez llegados al procesado de los datos será esta fase la encargada de realizar toda la lógica para la transformación, lectura, o cualquier procesado de los datos que se requiera.



- **Gestión datos de salida**: tras la finalización del procesado del mensaje, en caso de querer realizar el envío de los mensajes, existirá 4 formas totalmente iguales a la gestión de los datos de entrada para gestionar este envío. Por lo tanto, o será desde el mismo módulo de procesado desde donde se envíen los datos sin pasar por la fase de gestión de los datos de salida o podrán pasar a la sección de la gestión de los datos de salida, en el que otro hilo de ejecución se encargará de decidir el momento en el que se procederá a realizar este envío.



- **Envío de los datos (OutPut)**: como el nombre indica, será esta parte donde se realice el envío de los datos a todas las entidades que estén relacionadas a este MessageProcessor.

En los siguientes puntos se pasará a detallar en profundidad cada una de estas fases para facilitar su comprensión.

4.2.1 Entrada de los datos

La primera fase de un MessageProcessor, es la entrada de los datos, la cual puede ser externa al sistema, o interna recibiendo los datos desde los diferentes MessageProcessor que formen el sistema.

Para conseguir una gran variedad en la recepción de los datos de entrada, será posible la configuración de los MessageProcessors para ser capaces de interpretar los datos recibidos y conseguir de esta manera un elemento de interconexión entre el sistema y la entidad externa.

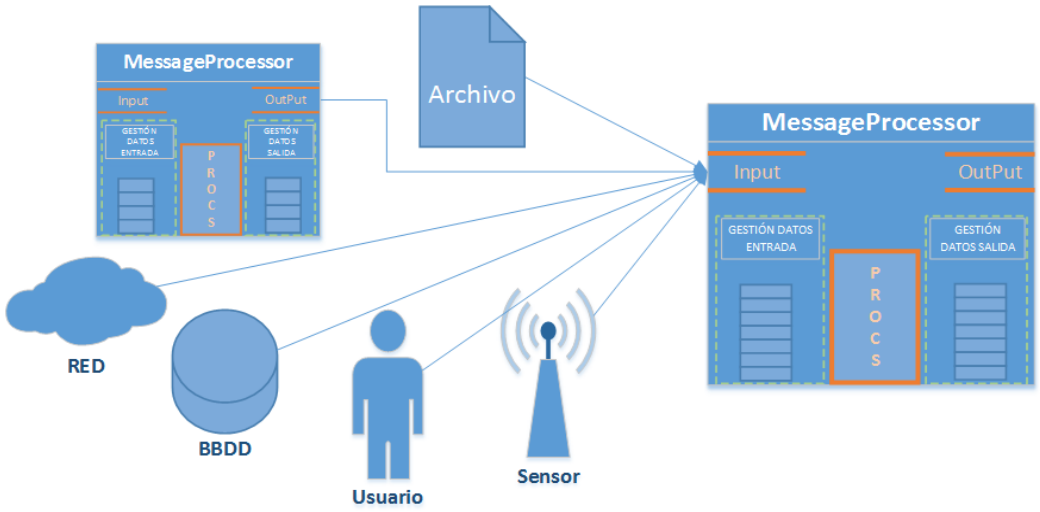


Ilustración 7. Datos de Entrada

En la Ilustración 7 podemos ver algunos ejemplos de los diferentes elementos a los que se podría adaptar los MessageProcessor para recibir los datos de entrada.

Aparte de destacar la posibilidad de recibir los datos desde entidades externas, la situación más común con la que nos encontraremos en el sistema, será la recepción de los datos desde la salida generada por otro MessageProcessor. Como se contará más adelante, esta conexión se realiza gracias al uso de una interfaz que facilita esta interconexión.

4.2.2 Gestión de los datos de entrada

La segunda fase del MessageProcessor no siempre tiene porque ser la misma, en este momento tras la recepción del mensaje pueden ocurrir dos cosas en base a la configuración que establezcamos.

Por un lado, tenemos la opción de procesar los datos de forma **síncrona,** donde se procesarán los datos según son recibidos, siendo el mismo hilo de ejecución que está enviando el dato el encargado de realizarlo.

Y por otro lado, está la posibilidad de realizar el procesamiento según las necesidades del problema, donde delegaremos esta funcionalidad en otro hilo de ejecución, consiguiendo de esta manera no bloquear el hilo que recibe los datos hasta que se haya terminado el procesado de la información y el posible envío de los mensajes generados por la entidad. Los métodos que permiten este tipo de gestión son; **asíncronos**, **por eventos** y **por tiempo**.

Una vez comprendido que existen diferentes formas de tratar los datos de entrada, se pasará a detallar cada una de estas posibilidades explicando las ventajas de cada una de ellas y el motivo por el cual se ha decidido facilitar estas funcionalidades en el propio framework.

4.2.2.1 Síncrono

Este tipo de gestión de los datos no entra dentro del módulo de gestión de datos de entrada, es el propio MessageProcessor el que se encarga de realizar esta gestión de los datos.

La forma síncrona en la gestión de los datos de entrada se ha establecido para tareas que no tengan una carga de trabajo elevada, en la cual, sea posible realizar las tareas de forma continua en un tiempo de respuesta reducido. Donde el tiempo, asegure que no ocurrirá un bloqueo en la entrada de los datos por un largo procesado de los mismos. Es decir, este método está pensado y orientado para aquellas funciones donde el tiempo de respuesta del procesado sea menor que la recepción de los datos o para aquellos casos, donde interese configurar una cadena de procesamiento síncrona, asegurando a las entidades el no enviar más datos hasta que no se hayan procesado los mensajes previos.

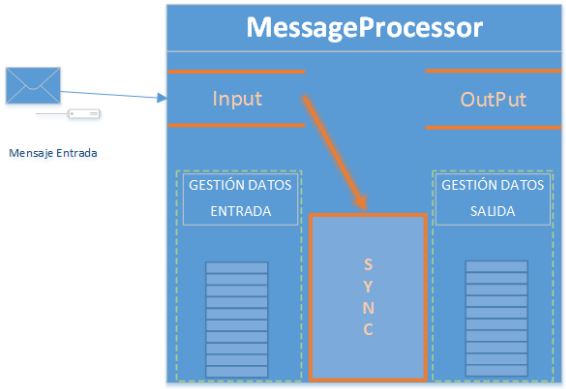


Ilustración 8. Gestión de los datos Síncrona

En la Ilustración 8 se puede apreciar como el mensaje no pasa a la fase de gestión de los datos de entrada, sino que pasarán a la fase de procesado directamente. Otro aspecto que se puede destacar de nuevo, es que debido a no usar un hilo para la gestión de los datos de entrada, no se podrá recibir un mensaje nuevo hasta el momento que se haya terminado el procesado del mensaje anterior.

4.2.2.2 Asíncrono

El método asíncrono sí que entra en juego con la fase de gestión de los datos. Esta gestión de los datos realiza el procesado según recibe los mensajes, al igual que la gestión síncrona. Pero existe una gran diferencia que nos proporcionará grandes beneficios. Esta diferencia, radica en la delegación de la gestión de los datos de entrada en un hilo de ejecución distinto.

Este nuevo hilo se encargará de ir almacenando en una cola o buffer los mensajes que van llegando y no pueden pasar a ser procesados por estar esta fase ocupada con el procesado de otro mensaje.

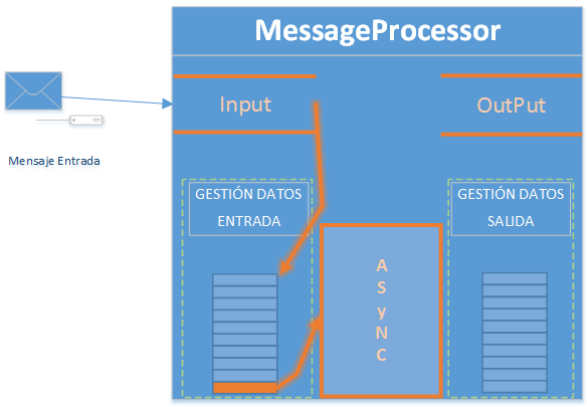


Ilustración 9. Gestión de los datos Asíncrona

En la Ilustración 9 se puede apreciar como a diferencia de la gestión síncrona, los mensajes no van directamente a la fase de procesado, sino que será en la fase de gestión de los datos de entrada, donde estarán los mensajes almacenados y se los irá proporcionando a la fase de procesado según vaya terminando de procesar del mensaje anterior.

4.2.2.3 Por Eventos

La gestión de los datos por eventos, nos permiten realizar procesamientos en base a una serie de eventos previamente configurados, los cuales, serán los que activen o indiquen el momento en el que los datos deben ser facilitados a la fase de procesamiento.

Como se puede deducir, hasta el momento en el que el evento se manifieste, el hilo encargado de la gestión de los datos de entrada, irá almacenando en una cola los mensajes que vayan llegando para posteriormente, en el momento que llegue el evento, pasar todos los mensajes almacenados a la fase de procesado.

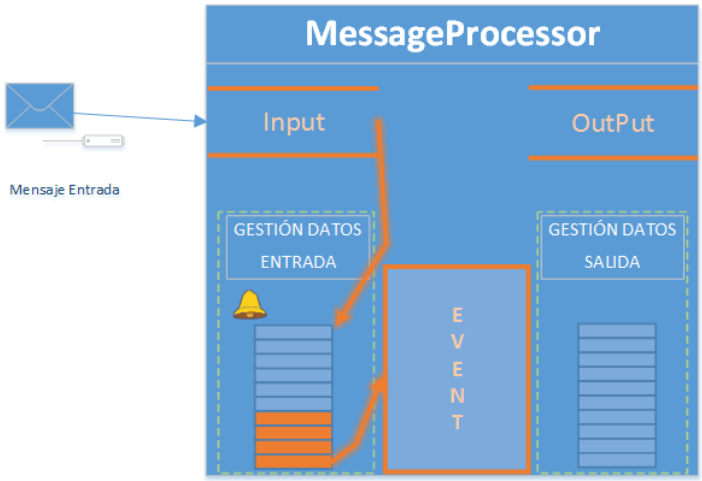


Ilustración 10. Gestión de los datos Por Eventos

Se puede ver en la Ilustración 10 que el funcionamiento de este MessageProcessor es igual que el asíncrono, la diferencia está en el momento que se proporcionan los datos a la fase de procesamiento.

Este tipo de gestión de los datos se configurará siempre que se quiera procesar un conjunto de datos cuando un evento ocurra. Un ejemplo que se puede relacionar con el proyecto, pudiera ser el procesado de todas las detecciones generadas por un sensor tras el paso por norte, con esto, garantizaremos que los datos que se van a procesar, corresponden a la detección de la misma vuelta del radar.

4.2.2.4 Por tiempo

En este tipo de gestión de los datos no se entrará en mucha profundidad, pues el concepto es el mismo reflejado en la gestión de los datos por eventos, la diferencia que debemos destacar, es que en vez de facilitar los datos a la fase de procesado por un evento que ocurra en el sistema, se realizará tras una franja de tiempo que se configure.

Los beneficios con los que nos podemos encontrar son los mismos, donde se consigue que la entrada de los mensajes no pueda ser bloqueada por un largo procesado de los datos.

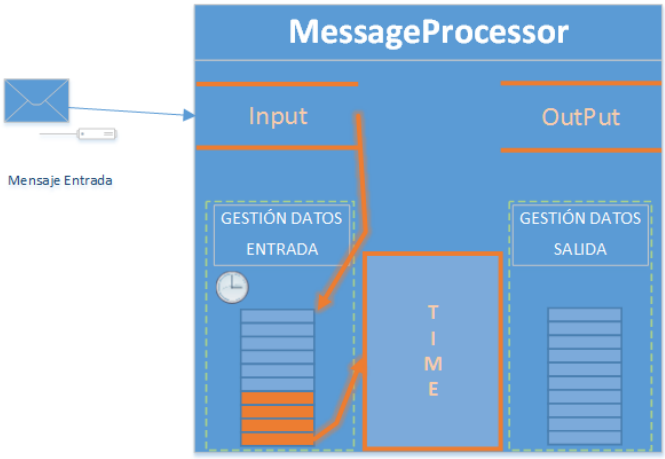


Ilustración 11. Gestión de los datos Por Tiempo

Esta Ilustración 11 muestra cómo se representará a lo largo del documento la gestión de los datos por tiempo, donde se representará a través del reloj en su fase y provocará que se realice el procesado de los datos por tiempos, indicado en la caja de procesado el texto **"TIME".**

4.2.3 Procesamiento de los datos

En la tercera fase nos encontramos con el módulo de procesado, el cual se encargará de realizar toda la lógica en el tratamiento de los datos. Será en este punto donde se configure la funcionalidad que debe realizar el MessageProcessor para llevar a cabo los objetivos del sistema.

Como se ha podido ver en la sección anterior, será posible ajustar la manera en la que se procesarán estos datos. Dependiendo de los objetivos del MessageProcessor, puede ser de mayor interés recibir los datos en conjunto para realizar el tratamiento de los mismos y sacar un resultado, o por el contrario, puede ser mejor ir procesando los datos según llegan los mensajes.

Otro aspecto a destacar en el procesado de los datos, es el tratamiento que se puede aplicar tras la finalización del procesado. Este tendrá la posibilidad de enviar los datos directamente o por el contrario, tendrá el mismo tipo de gestión de los datos para el envío, que la gestión de los datos de entrada, donde podrá almacenar los mensajes procesados en una cola de un hilo distinto para posteriormente ser enviados en base a las necesidades.

Se podría decir que las demás fases se centran en establecer la configuración para la comunicación y la gestión de los datos, y esta fase se centra en resolver los diferente problemas de procesado que tenga el sistema que lo implemente.

4.2.4 Gestión de los datos de Salida

Para la gestión de los datos de salida, se ha diseñado la posibilidad de configurarlo en base a las necesidades del problema. Se ha contemplado como posibles opciones el envío síncrono, asíncrono, por eventos o por tiempo.

Cabe destacar, que la gestión de los datos de salida es algo completamente independiente a la gestión de los datos de entrada que se haya configurado en las fases anteriores. La función que cumple la gestión de los datos de salida es la de proporcionar al MessageProcessor un hilo de ejecución, excepto en la gestión síncrona, donde almacenar los mensajes procesados y poder configurar el envío que se le quiere aplicar, permitiendo a la fase de procesado una mayor libertad para realizar los próximos procesados y no bloquearlo de esta manera hasta ser posible el envío del mensaje.

Por lo tanto, la fase de gestión de los datos de salida se refiere a la gestión del momento en el que los datos procesados pueden ser emitidos al siguiente MessageProcessor.

Al igual que en la gestión de los datos de entrada, y de forma totalmente análoga, nos encontramos con 4 posibles tipos de envío de los datos. Para ello, se podrá configurar de igual manera, los envíos asíncronos, por activación de un evento o por tiempos. Estos tipos envíos, son controlados por la fase de gestión de los datos de salida, sin embargo, como en la gestión de los datos de entrada, existe la posibilidad de realizar el envío de los datos de forma síncrona, donde será desde la fase de procesado desde la cual se realice este envío.

4.2.4.1 Síncrono

En primer lugar, se detallará la gestión de los datos de salida de forma síncrona, puesto que es la forma de gestión que se realiza de una forma distinta a las demás posibilidades. La gestión de los datos de salida de forma síncrona implica que según se quiera realizar un envío, el propio MessageProcessor, desde la fase de procesado será el encargado de transmitir los resultados a los destinos que estén establecidos.

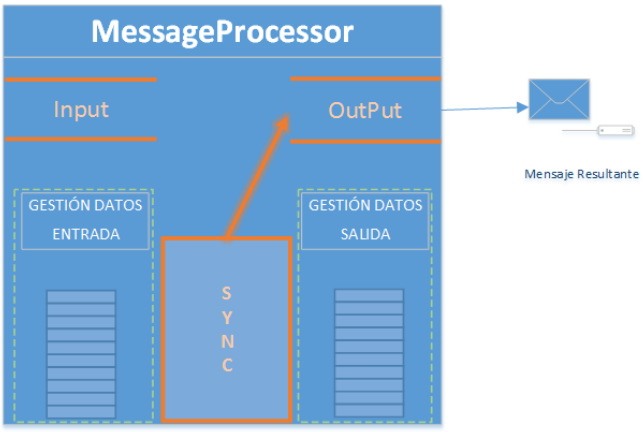


Ilustración 12. Gestión de los datos de salida Síncrono

Como se puede observar en la Ilustración 12, una vez realizado el procesado correspondiente, en el momento que se desee generar un envío, el propio MessageProcessor desde la fase de procesado, realizará el envío sin pasar por la fase de gestión de los datos de salida.

4.2.4.2 Asíncrono

Para el envío de los datos de forma asíncrona se utilizará otro hilo de ejecución, que será el encargado de enviar los datos en cuanto el MessageProcessor haya terminado de procesar los datos y llegue el momento de realizar el envío. El envío se produce en el mismo instante que en la forma síncrona, pero con la peculiaridad de realizarlo en otro hilo de ejecución.

Esto, proporciona un mayor rendimiento en las situaciones en el que se debe realizar grandes cantidades de información para ser procesada, por lo que delegando la transmisión de los datos a otro hilo, conseguimos que el MessageProcessor centre su ejecución en el procesado y no interrumpa su ejecución para realizar el envío de los datos.

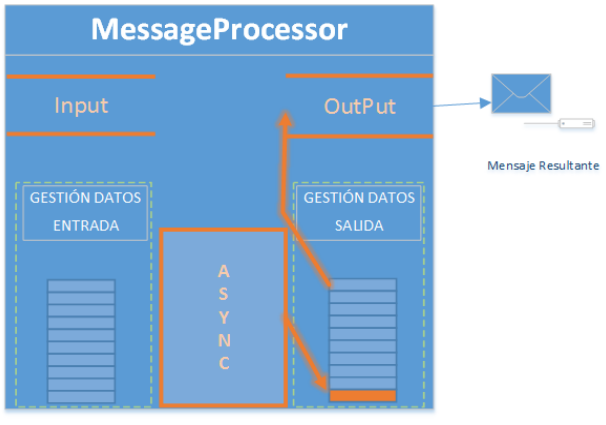


Ilustración 13. Gestión de los datos de salida Asíncrono

Se puede comprobar en la Ilustración 13, como en este caso, los mensajes a enviar pasan previamente por la gestión de los datos de salida para ser enviados en la fase de envío. Al ser un envío asíncrono, el envío se realizará en el mismo instante en el que la fase de procesado indique que se debe realizar el envío.

4.2.4.3 Por Eventos

Otra de las posibles formas de envío de los datos, es la programada por eventos, con este método, podemos fijar en qué momento o bajo qué circunstancias se activa el envió de los datos. El funcionamiento es totalmente reflejo a la gestión de los datos de entrada por eventos, lo que significa, que este método se ejecuta en otro hilo de ejecución y se va almacenando en una cola todos los datos listos para ser enviados de ese MessageProcessor. En el momento que el evento sea activado, enviará todos los datos de la cola a la vez.

Con este tipo de envío de los datos, podemos conseguir una configuración específica que requieran los diferentes MessageProcessor del sistema. En algunas circunstancias, es necesario un conjunto de datos para realizar el procesado o el tratamiento de los datos y resulta más eficiente, recibir estos datos al mismo tiempo en vez de recibirlos uno a uno, provocando un bloqueo en el siguiente MessageProcessor mientras está a la espera de todos los datos.

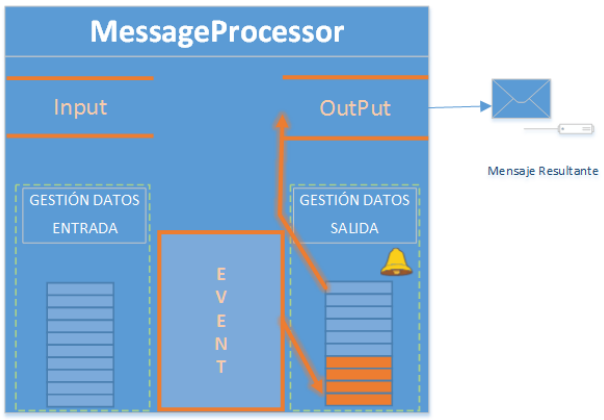


Ilustración 14. Gestión de los datos de salida Por Eventos

En la Ilustración 14, se puede ver la forma en la que se representará a lo largo del documento la configuración de la gestión de los datos de salida por eventos, donde la campana representará que se está a la espera de realizar un envío de los datos en base a un evento.

4.2.4.4 Por tiempo

Por último, nos encontramos con la gestión de los datos de salida por tiempo, este método es totalmente igual que el explicado anteriormente, por eventos. Esta forma de en la gestión de los datos de salida, también se realiza en otro hilo de ejecución y se almacena los datos procesados en una cola, la diferencia radica en que en vez de ser un evento el que active el envío de los datos, será una franja de tiempo configurable, la que establezca cada cuanto tiempo el MessageProcessor debe enviar los datos que tenga, en caso de no disponer de ningún mensaje para ser enviado, no se procederá al envío de ningún mensaje.

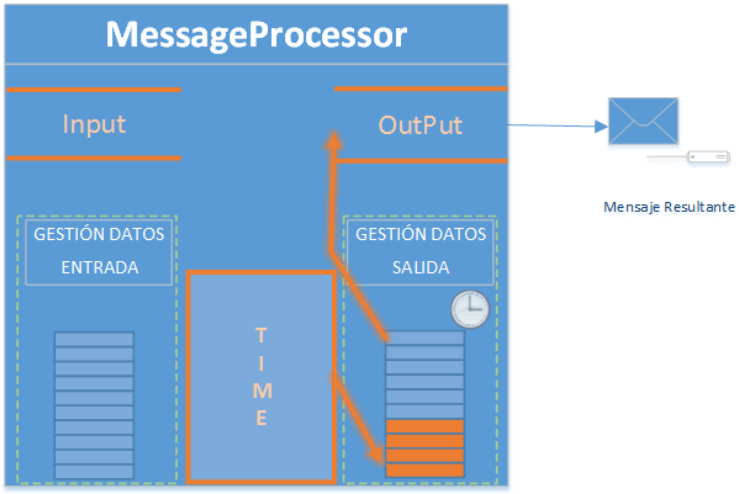


Ilustración 15. Gestión de los datos de salida Por Tiempo

Para representar la gestión de los datos de salida por tiempo, se hará uso de la Ilustración 15, donde mediante el reloj se reflejará este tipo de gestión en los datos de salida.

4.2.5 Envío de los datos

Por último, nos encontramos con la fase del envío de los datos tras el paso por las fases anteriores. Esta fase será la encargada de realizar el envío, para ello, se ha llevado a cabo una configuración en el MessageProcessor a través de una biblioteca de Boost conocida con el nombre de signals. Esta biblioteca, nos proporciona la funcionalidad de realizar las conexiones entre los diferentes MessageProcessor.

Llegados al momento en el que se produzca el envío, se realizará el envío a todos y cada uno de los MessageProcessor que estén conectados a la salida de este MessageProcessor, pues no hay forma de distinguir a quien se desea realizar el envío. Tendrá que ser desde el MessageProcessor que recibe el mensaje, desde donde mediante el tipo de mensaje, se detecte si es capaz de procesar el mensaje o por el contrario lo descarta sin tratarlo.

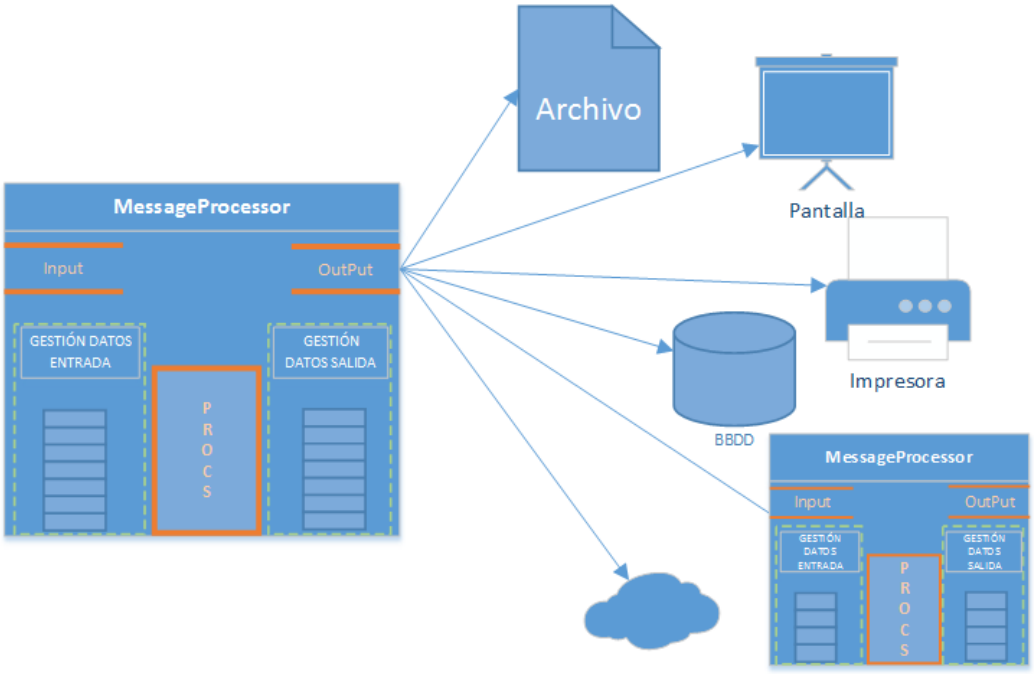


Ilustración 16. Envío de Mensajes MessageProcessor

En la Ilustración 16, se pretende demostrar todos los posibles tipos de MessageProcessor que se podrían especializar para realizar el envío a ese tipo de entidades.

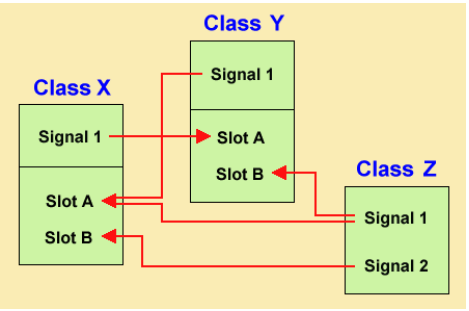
4.3 Tecnología utilizada

Como se ha comentado en el Estado del arte, el Framework se ha desarrollado utilizando C++ (C++), que es un lenguaje programación orientado a objetos y multiparadigma. C++ nos permite crear programas potentes y robustos destinados a la creación de sistemas complejos. Es un lenguaje muy extendido y dispone de una enorme fuente de información y documentación.

Además del propio lenguaje de programación C++, se ha utilizado una biblioteca muy conocida y extendida llamada **Boost**. Boost es un repositorio de bibliotecas multiplataforma que sirve para extender las capacidades del lenguaje C++. La utilidad de las bibliotecas Boost brillan por su facilidad de uso, ya que la inmensa mayoría de las mismas están totalmente definidas en ficheros de cabecera que simplemente incluiremos en nuestro proyecto, evitando tener que compilarlas y enlazarlas de forma independiente

Estas bibliotecas son de software libre, lo que supone que tengan una licencia de tipo GLP (General Public License), siendo muy parecidas a las licencias BSD y permitiendo ser utilizadas tanto en proyectos comerciales o no.

Dentro de las bibliotecas de Boost, se ha utilizado en el framework para la comunicación de las entidades, la biblioteca **Boost_Signals**. Esta biblioteca proporciona la implementación del mantenimiento de las señales del sistema. Los mensajes son enviados a los "slots", receptores que reciben los eventos o llamadas cuando una señal es emitida. Una de las principales ventajas que nos ofrece esta biblioteca, es la posible desconexión automática que ofrece definiendo el tipo de señal como **trackable**. Esta funcionalidad nos ayuda al mantenimiento entre las conexiones signal/slot, evitando que si un slot se borra, también se elimine su conexión, y así evitar problemas al llamar un slot inexistente.



(Elmue)

Ilustración 17. Representación Signals Slots

Como se representa en la Ilustración 17, la comunicación entre las diferentes clases son totalmente flexible con la utilización de signals y slots, en ella podemos observar, como al tener una misma señal más de un destino, emite automáticamente los mensajes a los diferentes MessageProcessor que tenga conectado.

Otra de las bibliotecas de Boost usadas en el Framework, es Thread, que nos permite la gestión hilos o threads, facilitando su creación y sincronización de forma independiente para las diferentes plataformas o sistemas operativos donde vaya a ejecutarse el MessageProcessor

Para la compartición de datos se ha utilizado la clase **shared_ptr,** es una plantilla que almacena el puntero sobre un objeto dinámico. Con ello, se intentan prevenir las pérdidas de memoria, liberando automáticamente los recursos: cuando un puntero (o el último de una serie de punteros) a un objeto es destruido. Estas características adicionales tienen como objetivo reducir errores causados por el mal uso de punteros, manteniendo la eficiencia.

Para el almacenamiento de una estructura de datos sobre ficheros de texto, binarios, en formato XML o cualquier soporte, se ha utilizado la biblioteca **Serialization** de Boost.

4.4 Diseño Entidades

Llegados a este punto tras haber planteado el problema y mostrado el diseño de la solución que se llevará a cabo, se adjuntará un diagrama para facilitar las partes que componen al MessageProcessor.

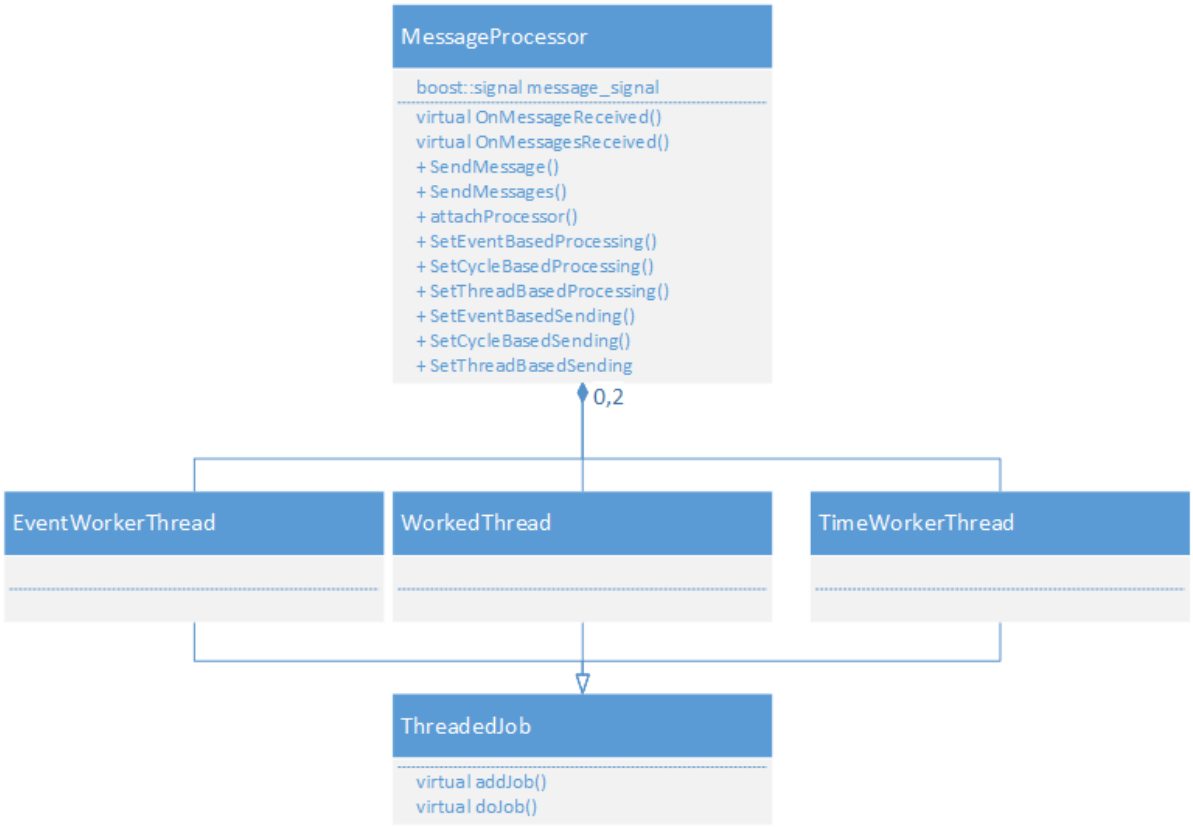


Ilustración 18. Diagrama de Clases

Se puede ver en la Ilustración 18 un diagrama de clases a alto nivel, donde se facilita las relaciones y funciones principales de las que consta el MessageProcessor. En él, se observa como el MessageProcessor en su creación, cuenta con una serie de métodos para establecer la configuración del mismo y cómo existe la posibilidad de estar compuesto de cero a dos clases que heredan de la clase *ThreadedJob*.

Estas dos posibles instancias, corresponderán a las dos fases opcionales que proporciona la entidad MessageProcessor, las cuales, son la gestión de los datos de entrada y la gestión de los datos de salida. Estas clases se crearán en base a la configuración que se le establezca, es por ello que podemos ver que esta clase puede ser de 3 tipos; *EventWorkerThread* que corresponde con una configuración por eventos, *WorkedThread* que será una configuración asíncrona y por último *TimeWorkerThread* que será la utilizada para la configuración por tiempo.

Conocida la estructura y los componentes del MessageProcessor, es importante mencionar que es una clase que está pensada para ser heredada y obtener de esta manera, las funcionalidades de comunicación que proporciona, siendo este, el objetivo principal del framework. La utilización de esta clase, ofrecerá la posibilidad de establecer diferentes tipos de comunicaciones y gestiones de los datos, todo ello, sin tener que configurarlo. Simplemente, deberemos establecer el tipo que se desea utilizar. También proporcionará la gestión de los mensajes que se manejan y por supuesto un eficiente paralelismo en el sistema.

4.4.1 MessageProcessor

En primer lugar se mostrará un uso típico de un MessageProcessor a través de su herencia.

```
class a : public MessageProcessor{
public:
  a(){}
  virtual ~a(){}
  virtual void processIncomingMessage(shared_message msg){}
  virtual void processIncomingMessages(const std::vector<shared_message>&shared_messages){}
};
```

Código 1. Clase MessageProcessor

En el Código 1, vemos como para la creación de una nueva clase que herede MessageProcessor, solo será necesario la implementación de dos métodos virtuales. Estos métodos serán processIncomingMessage() y processIncomingMessages(). En los siguientes puntos, se comentará las funciones de estos métodos con más detalle, pero su principal función, será la implementación del procesamiento que requiera realizar el MessageProcessor.

A continuación, se detallará los principales métodos que nos permitirán establecer la configuración del MessageProcessor una vez creada la clase y poder así llegar a crear la configuración que se precise.

4.4.1.1 Métodos para la Configuración de un MessageProcessor

En caso de no desear la configuración por defecto que ofrece el MessageProcessor, se podrá utilizar los siguientes métodos que mostrarán, mediante los cuales, se conseguirá establecer la configuración que se requiera y la implementación de la funcionalidad que deba realizar el mismo.

```
virtual void processIncomingMessage(shared_message) = 0;
virtual void processIncomingMessages(const std::vector<shared_message>&);
void sendMessage(Message*);
void sendMessages(std::vector<Message>&);
void attachProcessor(MessageProcessor*);
bool setEventBasedProcessing(std::vector<MessageType>&);
bool setCycleBasedProcessing(unsigned long milliseconds);
bool setThreadBasedProcessing();
bool setEventBasedSending(std::vector<MessageType>&);
bool setCycleBasedSending(unsigned long milliseconds);
bool setThreadBasedSending();
processor_id getProcessorId() const;
processor_config& getConfig();
```

Código 2. Métodos Configuración MessageProcessor

En la tabla del Código 2 vemos la interfaz de los métodos que se explicarán a continuación especificando su función y la manera de usarlo.

4.4.1.1.1 AttachProcessor(MessageProcessor*)

El método AttachProcessor, servirá para añadir aquellos MessageProcessor a los cuales, tras realizar la acción de envió, les llegue nuestro mensaje.

En este método será donde se implemente las relaciones que existan entre los diferentes MessageProcessor a través de los signal y slots detallados en la sección Tecnología utilizada. Tal y como se ha explicado, los signals corresponderán a las señales encargadas de emitir a todos aquellos slots que estén enlazados.

Para llevar a cabo este enlace, se puede ver en los parámetros que recibe el método, como se le debe pasar el puntero del MessageProcessor al que se debe relacionar, con esto, conseguiremos que reciba todos los mensajes emitidos por este.

```
a* messageProcessor1 = new a();
a* messageProcessor2 = new a();
messageProcessor1->attachProcessor(messageProcessor2);
```

Código 3. Uso AttachProcessor()

En el ejemplo del Código 3, podemos ver lo simple que será llevar a cabo este enlace. Tras la creación de dos MessageProcessor de la clase “a” especificada en el Código 1, vemos como el MessageProcessor1, en el método AttachProcessor, le pasar por parámetro el MessageProcessor2. A partir de este momento, todos los mensajes enviados por el MessageProcessor1, serán recibidos en el MessageProcessor2.

4.4.1.1.2 SetEventBasedProcessing(std::vector<MessageType>&)

Este método servirá para especificar al MessageProcessor una gestión en la entrada de los datos por eventos, la cual, procesará los datos como se ha detallado en la sección 4.2.2.3 Por Eventos.

```
vector<MessageType> myEvents;
myEvents.push_back(RADAR_NORTH_EVENT);
messageProcessor1->setEventBasedProcessing(myEvents);
```

Código 4. Uso SetEventBasedProcessing()

Como vemos en el Código 4, a la hora de asignar la configuración de la gestión de la entrada de los datos por eventos, será necesario pasarle por parámetro un vector con los tipos de mensajes que activarán el momento del procesado tal y como se ha detallado en la sección 4.2.2.3 Por Eventos.

En el ejemplo podemos ver como se crea el vector del tipo MessageType y a continuación, como se le añade el tipo de mensaje RADAR_NORTH_EVENT.

4.4.1.1.3 SetCycleBasedProcesing(unsigned long milliseconds)

Este método servirá para especificar al MessageProcessor una gestión en la entrada de los datos por tiempo.

```
messageProcessor1->setCycleBasedProcessing(2000);
```

Código 5. Uso SetCycleBasedProcesing

Vemos en el Código 5, como para establecer esta configuración, se le deberá especificar en milisegundos, el tiempo que se desea fijar, para realizar el procesado tras el paso de esa franja de tiempo.

4.4.1.1.4 SetThreadBasedProcesing()

Este método servirá para especificar al MessageProcessor una gestión en la entrada de los datos asíncrona.

```
messageProcessor1->setThreadBasedProcessing();
```

Código 6. Uso SetThreadBasedProcesing

Se observa en el Código 6, que para realizar esta configuración, solo se tendrá que llamar al método y no será necesario ningún parámetro, pues el procesado que se llevará a cabo será tal y como se ha especificado en la sección 4.2.2.2 Asíncrono.

4.4.1.1.5 SetEventBasedSending(std::vector<MessageType>&))

Este método servirá para especificar al MessageProcessor una gestión en la salida de los datos por eventos.

```
vector<MessageType> myEvents;
myEvents.push_back(RADAR_NORTH_EVENT);
messageProcessor1->setEventBasedSending(myEvents);
```

Código 7. Uso SetEventBasedSending

Como vemos en Código 7, el uso de este método es igual que en la gestión en los datos de entrada por eventos. Donde con la especificación de los eventos que activarán el envío, será más que suficiente para poder disponer de esta configuración.

4.4.1.1.6 SetCycleBasedSending(unsigned long milliseconds)

Este método servirá para especificar al MessageProcessor una gestión en la salida de los datos por tiempo.

```
messageProcessor1->setCycleBasedSending(1000);
```

Código 8. Uso SetCycleBasedSending

Como vemos en el Código 8, el uso es totalmente igual que en la gestión de los datos de entrada por tiempo. Solamente, se requerirá especificar el tiempo que debe pasar, para realizar los envíos.

4.4.1.1.7 SetThreadBasedSending()

Este método servirá para especificar al MessageProcessor una gestión en la salida de los datos asíncrono.

```
messageProcessor1->setThreadBasedSending();
```

Código 9. Uso SetThreadBasedSending

Como se puede ver en el Código 9, el uso de este método es igual que el de la gestión de los datos de entrada asíncrona. Con la llamada al método será más que suficiente, para que el framework internamente, se encargue de realizar este tipo de gestión de los datos de salida.

4.4.1.1.8 SendMessage(Message*)

Mediante el método sendMessage(), será desde donde el MessageProcessor tendrá la opción de enviar aquellos mensajes que requiera enviar. Para ello, solo tendrá que pasar por parámetro el puntero del mensaje a enviar.

Para especificar un poco más en el funcionamiento interno del método, tras realizar la llamada al método, es importante mencionar que la referencia de este mensaje pasará a ser un mensaje del tipo **shared_ptr.** Como se ha comentado en la Tecnología utilizada, este tipo de datos nos proporcionará la gestión de la memoria mediante el borrado de la variable en el momento que no se esté usando por ningún elemento del sistema.

Por lo tanto, todos los mensajes que se manejen en el sistema, serán convertidos por el framework automáticamente a este tipo de datos, siendo una de las grandes ventajas que suministra el framework con su uso.

```
Message* sms = new Message();
        messageProcessor1->sendMessage(sms);
```

Código 10. Uso SendMessage

Haciendo uso de la clase “a” creada en el Código 1, vemos cómo tras la creación de un mensaje mediante el uso de método sendMessage(sms), pasándole por parámetro la dirección de memoria del mensaje, se llevará a cabo el envío que tengamos configurado, con los métodos anteriormente detallados.

Es importante destacar que el envío de los datos se realizará en base a la configuración que se le haya dado al MessageProcessor, por lo tanto, será solo en los casos de tener una configuración síncrona o asíncrona donde realmente se realice el envío en ese instante. En el caso de tener la configuración por evento o por tiempo se almacenará el mensaje en las colas correspondientes y se producirá el envío llegado el momento de la configuración.

4.4.1.1.9 ProcessIncomingMessage(shared_message), ProcessIncomingMessages(std:vector<shared_message>&)

Estos métodos, serán los que le aporte toda la funcionalidad al MessageProcessor, pues serán los encargados de ejecutarse en la fase de procesado.

El caso del método ProcessIncomingMessage(), será el método llamado cuando se tenga una configuración de los datos de entrada síncrona o asíncrona.

Para el caso en el que la configuración de los datos de entrada sea la de eventos o por tiempos, el método al que se llamará, es ProcessIncomingMessages(). El motivo por el cual existe una distinción entre los métodos de procesado, se debe simplemente al número de mensajes que recibirá cada uno de ellos, es por ello, que la diferencia radica en los parámetros recibidos, donde uno de ellos será un único mensaje y en el otro un vector de mensajes.

Quizás, esta parte ayude a comprender porque el procesado síncrono y asíncrono es exactamente igual, con la peculiaridad de tener una gestión en la entrada de los datos. El funcionamiento será el mismo, pero tendremos de por medio un hilo que se encargue de la recepción de los mensajes

No se proporcionará ejemplos de estos métodos puesto que simplemente se requiere el trato de los mensajes recibidos, ya sea uno a uno o con el vector de mensajes que se reciban.

4.4.2 ThreadedJob

Esta clase como se puede ver, tiene 3 diferentes implementaciones de la misma. Cada una de ellas, estará destinada para los diferentes tipos de gestión de los datos. Hay que destacar, que las mismas servirán tanto para la gestión de los datos de entrada como para la gestión de los datos de salida. Interiormente, el framework sabrá discernir si debe enviar los mensajes a la fase de procesado o a la de envío de los datos.

Como se ha mencionado a lo largo de las explicaciones de los diferentes métodos de gestión en la entrada y salida de los datos, estos se realizarán en un hilo diferente. La manera con la que se ha conseguido este objetivo es con la implementación de estas clases, las cuales, harán uso de la biblioteca thread de Boost para gestionarlo.

En esta clase, podemos destacar los dos métodos principales que puedan ser de interés a la hora de utilizar en framework. Como se ha visto en la Ilustración 18, tenemos los métodos AdJob() y DoJob(). Tal y como se puede intuir con el nombre de los métodos, AdJob se encargará de ir almacenando los mensajes en la cola propia del proceso y doJob se encargará de enviar los mensajes a la fase que corresponda.

4.5 Funcionamiento de MessageProcessor

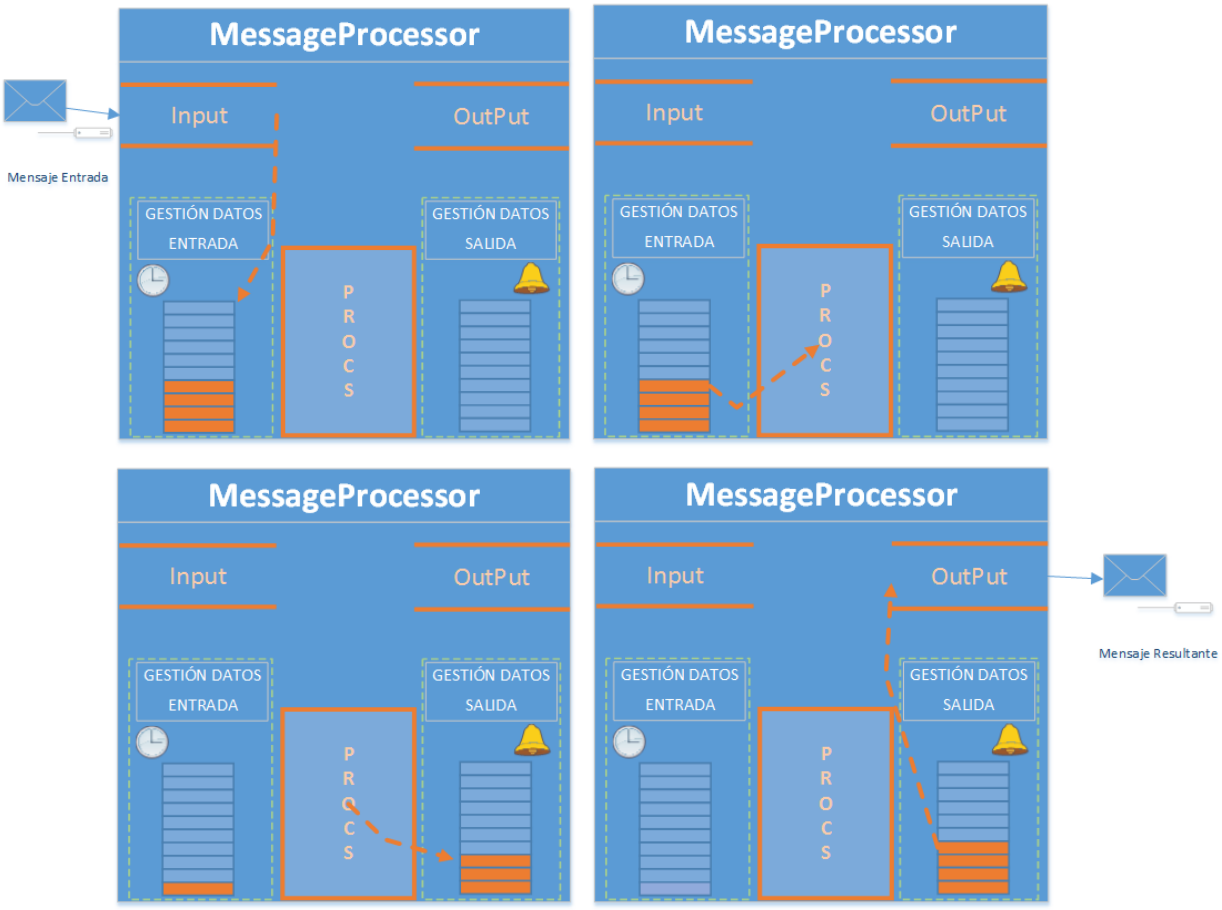


Ilustración 19. Funcionamiento MessageProcessor

En la representación de la Ilustración 19, se refleja cada uno de los pasos que realiza el MessageProcessor desde la recepción de un mensaje, hasta el envío de los mismos tras su procesado.

En primer lugar, destacaremos la configuración que tiene el MessageProcessor, esto se puede comprobar, en los iconos que aparecen en las secciones gestión de datos de entrada y en la gestión de los datos de salida. Como se puede observar, la configuración que se ha definido para la gestión de los datos de entrada, es por tiempo, y en la gestión de los datos de salida, se le ha definido una salida por eventos.

Tras conocer previamente como se va tratar los datos, detallaremos paso por paso su funcionamiento.

En la primera imagen se puede ver que el MessageProcessor recibe los datos desde otra entidad. Al tener configurado una gestión de los datos de entrada por tiempo, se

aprecia, cómo los mensajes que van llegando durante esa franja de tiempo estipulada, se van almacenando en la cola de un hilo de ejecución diferente al del propio MessageProcessor. Esta es la principal diferencia con la configuración síncrona, la cual no hace uso de esta fase y directamente los mensajes pasan a ser procesados.

En la segunda imagen, ya podemos ver como la franja de tiempo que tiene fijada el MessageProcessor se ha cumplido, y es en este momento, en el que la fase de la gestión de los datos de entrada va pasando los mensajes a la fase de procesamiento.

Ya en la tercera imagen, se representa cómo el MessageProcessor una vez procesa los mensajes, tiene dos opciones, en este caso en concreto al tener una configuración de la gestión de los datos de salida por evento, los mensajes procesados pasan a ser almacenados en una cola de un hilo distinto al del MessageProcessor, la otra opción que existiría en el caso de tener una configuración síncrona, sería la de realizar en la misma fase de procesamiento el envío del mensaje tras su procesado.

Para terminar, en la cuarta imagen, se aprecia el funcionamiento de la gestión de los datos de salida, donde llegado el momento del evento, esta fase se encargará de proporcionar los mensajes para ser enviados.

Para facilitar la visualización del MessageProcessor en diferentes situaciones, añadiremos pequeños ejemplos, donde podremos ver la configuración que se ha implementado y como los resultados, concuerdan con esta configuración. Además, para ver un poco de esta implementación en código, se adjuntará algunas secciones relevantes que permitan ver cómo es posible llevar a cabo este problema.

Funcionamiento MessageProcessor con gestión de los datos por eventos.

Se pretende demostrar el funcionamiento de un procesamiento por eventos, recibiendo 8 mensajes que hayan sido enviados desde otro MessageProcessor.

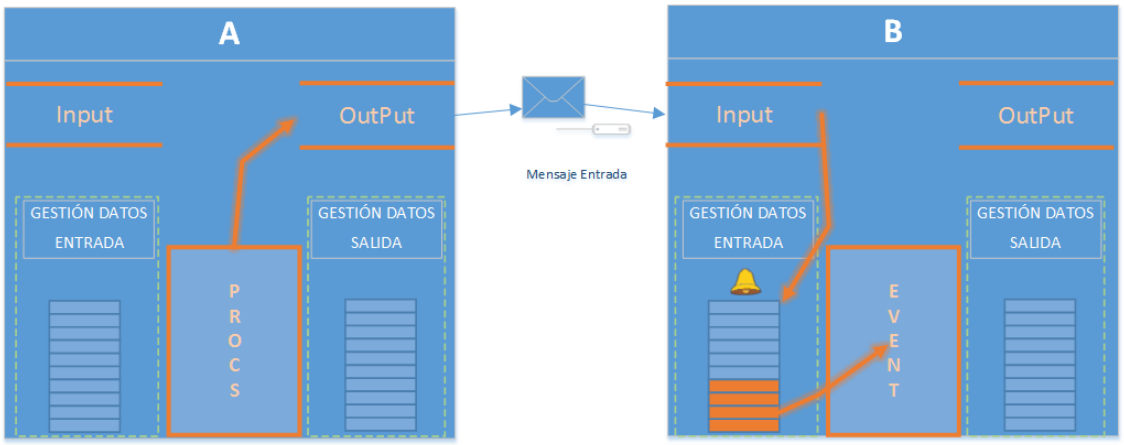


Ilustración 20. Funcionamiento MessageProcessor con gestión de los datos por eventos

En la Ilustración 20, se puede ver gráficamente el problema planteado.

A continuación, se adjunta un fragmento de código con el fin de poder comprobar la facilidad de llevar a cabo este problema, mediante la creación de dos clases que hereden de MessageProcessor.

Puesto que el MessageProcessor "a", no realiza ningún procesado sobre los datos y solamente envía mensajes, solo se mostrará el código del MessageProcessor "b", el cual contiene alguna configuración más.

```
class b : public MessageProcessor{
public:
	b(){
		vector<MessageType> myEvents;
		myEvents.push_back(RADAR_NORTH_EVENT);
		setEventBasedProcessing(myEvents);
	}
	virtual ~b(){}
	virtual void processIncomingMessage(shared_message msg)
	{
		cout << "B Recieve 1 message" << endl;
	}
	virtual void processIncomingMessages(const std::vector<shared_message>&
shared_messages){
		cout << "B Recieve " << shared_messages.size() << " messages" << endl;
	}
};
```

Código 11. Configuración clase B Por Eventos

En el código de la clase b proporcionado en el Código 11, podemos ver como en el constructor, especificamos el procesamiento por eventos mediante la función ***setEventBasedProcessing(myEvents);***, en la que le pasamos por parámetro, un vector de los posibles eventos que pueden activar el procesamiento de esta entidad.

Por lo demás, para realizar este ejemplo tan simple, solo tendremos que implementar el método `virtual void processIncomingMessage(shared_message msg)`,en el caso que el procesamiento sea síncrono o asíncrono, en el cual se procesarán los mensajes uno a uno según son recibidos, o por el contrario, se deberá configurar el método `virtual void processIncomingMessages(const std::vector<shared_message>& shared_messages)` que será, el que tenga la configuración del procesamiento en el caso de tener configurado un procesamiento por eventos o por tiempo, los cuales, procesarán el conjunto de mensajes que hayan recibido.

```
void Main ()
{
	a* claseA = new a();
	b* claseB = new b();
	claseA->attachProcessor(claseB);

	claseA->sendMessage(new Message());
	claseA->sendMessage(new Message());
	claseA->sendMessage(new Message());
	claseA->sendMessage(new Message());

	Message* evento = new Message();
	evento->setMessageType(RADAR_NORTH_EVENT);
	claseA->sendMessage(evento);

	claseA->sendMessage(new Message());
	claseA->sendMessage(new Message());
	claseA->sendMessage(new Message());
	claseA->sendMessage(new Message());

	Message* evento2 = new Message();
	evento2->setMessageType(RADAR_NORTH_EVENT);
	claseA->sendMessage(evento);

}}
```

Código 12. Main Funcionamiento Por Eventos

En el código facilitado del Código 12, podemos ver en el main la instancición de los dos MessageProcessor, “a” y “b”.

Lo siguiente que se puede ver, es como mediante el método ***attachProcessor(claseB);*** se relaciona los MessageProcessor, provocando que el

MessageProcessor "b", reciba todos los mensajes que el MessageProcessor "a" emita con el método ***sendMessage()***.

Finalmente, tras probar el ejemplo, veremos en los resultados como el MessageProcessor "a" enviará 4 mensajes y será en el 5º mensaje, donde envíe un evento para que el MessageProcessor pase a procesar los mensajes recibidos hasta el momento. A continuación, se volverá a provocar la misma situación volviendo a enviar 4 mensajes y un 5º que sea el evento donde le indicará al MessageProcessor "b" que realice el procesado.

```
A Send  messages -->
A Send  messages -->
A Send  messages -->
A Send  messages -->
A Send  Event -->
B Recieve 4 messages <--
A Send  messages -->
A Send  messages -->
A Send  messages -->
A Send  messages -->
A Send  Event -->
B Recieve 4 messages <--
```

Ilustración 21 . Resultado Funcionamiento MessageProcessor Por Eventos

En la Ilustración 21, podemos comprobar como el MessageProcessor "a" envía los mensajes y como hasta no llegar el momento del envío del evento, el MessageProcessor "b" no empieza a procesar.

Funcionamiento MessageProcessor Procesamiento por Tiempo

En este pequeño ejemplo se mostrará el funcionamiento del ejemplo anterior pero con la modificación en el MessageProcessor "b" el cual realizará el procesado por tiempo en vez de realizarlo por eventos.

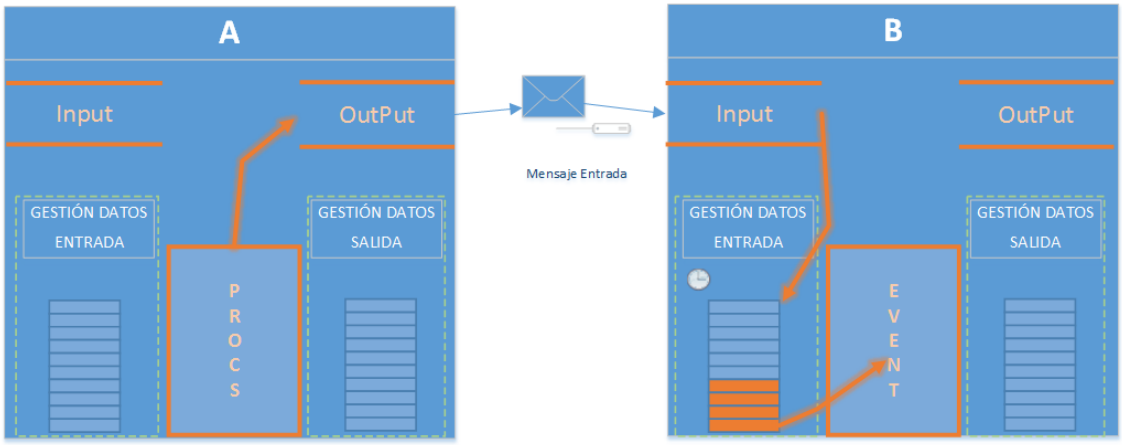


Ilustración 22. Funcionamiento MessageProcessor con gestión de los datos por tiempo

En la Ilustración 22, se puede comprobar gráficamente la configuración que tienen los MessageProcessor tal y como se ha explicado a lo largo de los apartados del framework de procesado de la información.

```
class b : public MessageProcessor{
public:
	b(){
		setCycleBasedProcessing(1000);
	}
	virtual ~b(){}
	virtual void processIncomingMessage(shared_message msg)
	{
		cout << "B Recieve 1 message" << endl;
	}
	virtual void processIncomingMessages(const std::vector<shared_message>&
shared_messages){
		cout << "B Recieve " << shared_messages.size() << " messages" << endl;
	}
};
```

Código 13. Configuración clase B Por Tiempo

En el Código 13, vemos que el MessageProcessor "b" es el mismo pero con la configuración en el constructor distinta, esto se debe a que en este caso procesará los mensajes recibidos por tiempo, para ello con el método **setCycleBasedProcessing(1000);** estamos configurando un procesamiento de los mensajes cada 1 segundo, este tiempo se fija pasándole por parámetro el tiempo en milisegundos que deseamos asignarle a cada rodaja de tiempo en la que deseamos que se procesen los mensajes.

```
void Main ()
{
        a* claseA = new a();
        b* claseB = new b();
        claseA->attachProcessor(claseB);

        claseA->sendMessage(new Message());
        claseA->sendMessage(new Message());
        claseA->sendMessage(new Message());
        claseA->sendMessage(new Message());
        claseA->sendMessage(new Message());
        claseA->sendMessage(new Message());
        claseA->sendMessage(new Message());
        claseA->sendMessage(new Message());
        claseA->sendMessage(new Message());
        claseA->sendMessage(new Message());
        claseA->sendMessage(new Message());
        claseA->sendMessage(new Message());
        claseA->sendMessage(new Message());
        claseA->sendMessage(new Message());


        return true;
}}
```

Código 14. Main Funcionamiento Por Tiempo

En este caso vemos como en el Código 14, el MessageProcessor "a" envía 12 mensajes sin necesidad de crear un evento para activar el procesamiento en el MessageProcessor "b".

```
A Send  messages -->
A Send  messages -->
A Send  messages -->
A Send  messages -->
A Send  messages -->
A Send  messages -->
A Send  messages -->
A Send  messages -->
A Send  messages -->
A Send  messages -->
A Send  messages -->
A Send  messages -->
B Recieve 12 messages <--
B Recieve 0 messages <--
B Recieve 0 messages <--
B Recieve 0 messages <--
B Recieve 0 messages <--
B Recieve 0 messages <--
B Recieve 0 messages <--
B Recieve 0 messages <--
```

Ilustración 23 . Resultado Funcionamiento MessageProcessor Por Tiempo

Tras la ejecución del programa se puede observar como los mensajes enviados por el MessageProcessor "a" son recibidos todos por el MessageProcessor “b” en el primer periodo de tiempo por lo que se procesan todos en la primera vez que transcurre la

franja de tiempo, al no seguir enviando mensajes el MessageProcessor "a", el MessageProcessor "b" mostrará cada 1 segundo el mensaje de que no ha recibido ningún mensaje ya que la configuración establecida provoca que continuamente tras el paso de la franja de tiempo intenta procesar los mensajes recibidos aun no teniendo ninguno en la cola.

Funcionamiento MessageProcessor Síncrono

Como último ejemplo, se mostrará el funcionamiento de un MessageProcessor que procese los mensajes de forma síncrona, con el fin de poder ver las mismas ejecuciones con las distintas posibilidades.

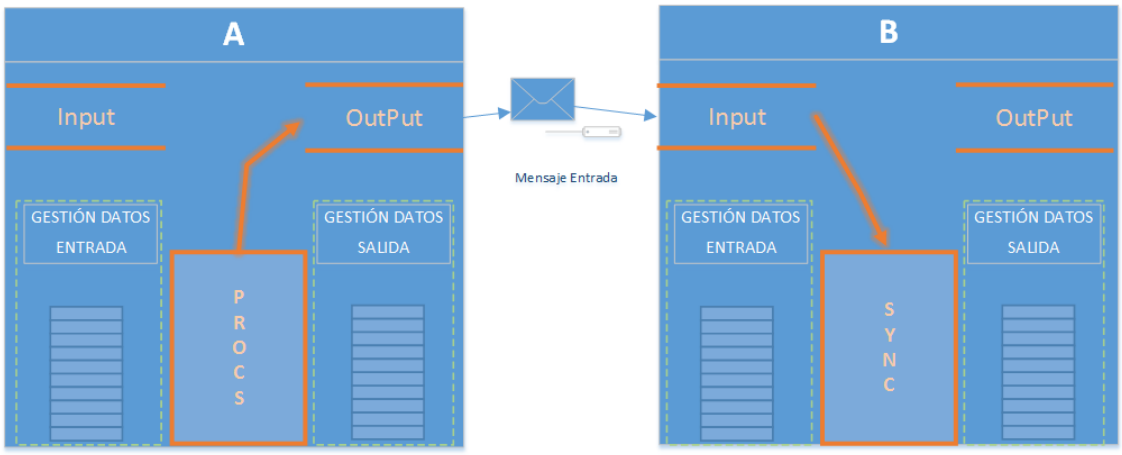


Ilustración 24. Funcionamiento MessageProcessor con gestión de los datos Síncrona

En la Ilustración 24, una vez más se puede ver gráficamente el problema que se pretende demostrar, donde se ve como el procesamiento de los datos se realiza de forma síncrona.

```
class b : public MessageProcessor{
public:
	b(){}
	virtual ~b(){}
	virtual void processIncomingMessage(shared_message msg){
		cout << "B Recieve 1 message" << endl;
	}
	virtual void processIncomingMessages(const std::vector<shared_message>&
shared_messages){
		cout << "B Recieve " << shared_messages.size() << " messages" << endl;
	}
};
```

Código 15. Configuración clase B Síncrono

Al ser un procesamiento síncrono el que tiene que realizar el MessageProcessor “b”, podemos ver en el Código 15, como la configuración que se le establece es la de por

defecto, ya que la clase MessageProcessor en su configuración inicial tiene la creación de un MessageProcessor síncrono, por lo tanto no será necesario configurar el tipo de procesado de los datos ni el envío de los mismos.

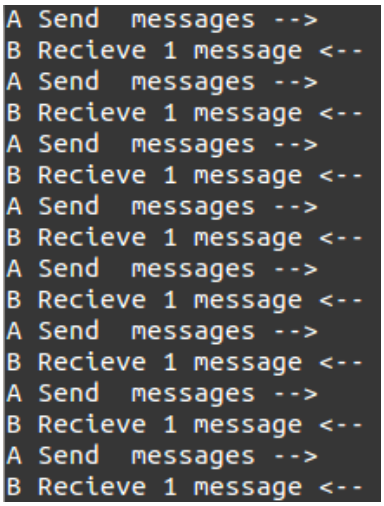


Ilustración 25 . Resultado Funcionamiento MessageProcessor Síncrono

En los resultados de la Ilustración 25 . Resultado Funcionamiento MessageProcessor , podemos ver como la configuración síncrona concuerda con lo esperado, donde el mensaje enviado por el MessageProcessor "a" hasta que no se finalice el procesado en el MessageProcessor "b", no pasará a enviar el siguiente mensaje, esto se debe a que es el mismo hilo de ejecución el encargado de realizarlo toda la acción.

Con este ejemplo podemos comprobar cómo un largo procesado de los datos en el MessageProcessor "b", podría llevarnos a colapsos en el MessageProcessor "a" y como el proporcionar la posibilidad de realizar los diferentes procesados y envíos nos permite tener un sistema mucho más eficiente que se adapta las necesidades.

4.6 Comunicación

En el Framework se ha diseñado para soportar una comunicación entre los módulos del sistema totalmente flexible. Con flexible, se quiere expresar la posibilidad de relacionar cualquier MessageProcessor del sistema, con uno o con varios, permitiendo cualquier tipo de configuración.

Una de las mayores ventajas que se consiguen con la utilización de este Framework, es la posibilidad de realizar las conexiones de las entidades con el sistema en ejecución, lo que supone, poder incluir cualquier MessageProcessor con una nueva funcionalidad en cualquier momento sin necesidad de parar el sistema para que esto ocurra.

Otra ventaja a destacar, es la de permitir a los MessageProcessor tener como entrada, la salida de varias entidades, y de igual manera, la posibilidad de enviar un mismo dato a la entrada de varios MessageProcessor, con esto, se consigue ampliar más aun la flexibilidad del sistema y permite proporcionar una escalabilidad sin límites.

Los tipos de conexiones con los que nos podemos encontrar son relaciones de uno a uno, uno a mucho o muchos a uno.

El MessageProcessor, gestiona automáticamente la problemática de las conexiones mediante signals y slots de la librería Boost. Esta librería además de las funciones de comunicación que nos proporciona, nos encontramos con que mediante el uso de esta librería también nos facilita la desconexión de aquellas relaciones que tengan establecidas entre MessageProcessors y desaparezcan del sistema, función que nos simplifica la programación del sistema, al no tener que llevar el control de estas relaciones al completo, eliminando de esta manera, los posibles errores que pudieran ocurrir en situaciones no controladas.

A continuación, podemos reflejar mediante un ejemplo las comunicaciones entre diferentes MessageProcessor del sistema y ver algunas de las posibilidades de interconexión que se pueden configurar.

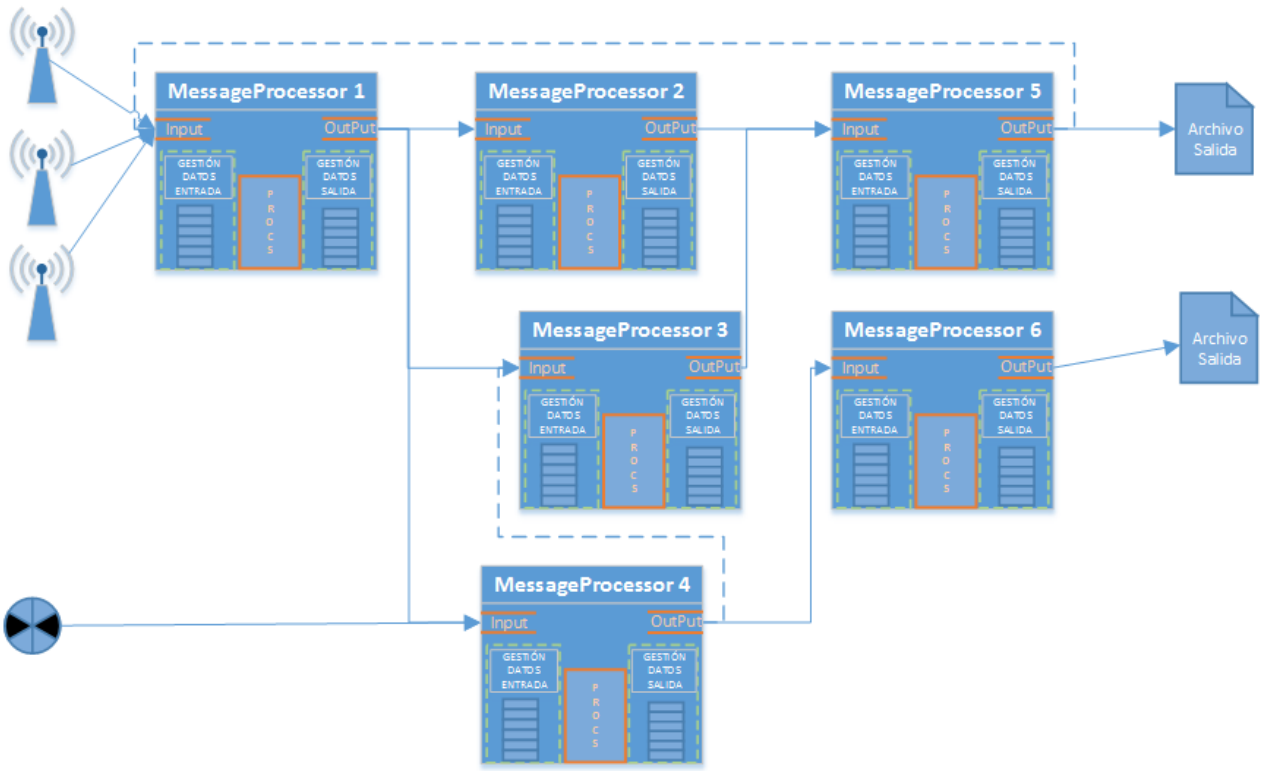


Ilustración 26. Comunicación entre MessageProcessor

Como se puede observar en la Ilustración 26, puede existir uno o varios MessageProcessor que se comporten como los receptores de los datos desde el exterior del sistema. En este ejemplo, el MessageProcessor 1 y el MessageProcessor 4 serán los que reciban datos desde sensores.

El MessageProcessor 1, puede simular un módulo el cual se encarga de filtrar los datos de entrada, por lo que implica, que simplemente su función será la de gestionar la distribución de los datos a los siguientes MessageProcessor conectados a él. En este caso hipotético, los mensajes producidos por el MessageProcessor 1, los recibirán los MessageProcessor 2, 3 y 4.

Otra característica que se puede apreciar en el ejemplo, es la posibilidad de transmitir los datos tras el procesado, donde estos resultados serán la entrada de otro MessageProcessor.

Finalmente, otro aspecto que podemos observar en la Ilustración 26 pudiera ser la salidas de los datos, estas salidas se pueden conectar y desconectar cuando sea necesario, pudiendo producir diferentes puntos de salida para el sistema. Estas salidas del sistema servirán para generar logs, archivos de incidencias en el caso de ocurrir algún problema, almacenar un tipo de procesado que representa un formato de datos específico, un formato de salida de los datos capaz de ser interpretado por programas

externos, pueden ser enviados por red a otro módulo conectado en otro punto del sistema, etc., lo que se pretende demostrar, es la posibilidad de que un mismo dato de salida pueda ser tratado simultáneamente por otros MessageProcessor dándole diferentes salidas.

En este caso en concreto, vemos como los MessageProcessor 5 y 6 pudieran estar especializados en la serialización de los datos que reciben sobre un fichero de salida.

Como conclusión de la comunicación entre los diferentes módulos del sistema, se puede decir que permite cualquier tipo de configuración que se requiera, que no tiene límites de interconexión, y la principal ventaja que proporciona, es la gestión eficiente de estas comunicaciones que pueden ser controladas sin bloqueos gracias a las funcionalidades que se han proporcionado en la fase de gestión de los datos de entrada y de la gestión de los datos de salida.

4.7 Utilidades

Con el fin de proporcionar al Framework una mayor funcionalidad, se han añadido algunas utilidades de gran interés que facilitará la representación, almacenamiento y transmisión de los datos tanto en el sistema como en entidades externas.

Algunos de los complementos añadidos al framework son el Recorder y Player, con los cuales nos permitirán grabar datos sobre ficheros y leerlos para poder volver a reproducirlos. Otra utilidad que se ha añadido, es la posibilidad de configurar el MessageProcessor para comunicarse mediante sockets.

4.7.1 Recorder

Recorder es una utilidad que se ha creado para dar la posibilidad de serializar los datos en un fichero. Con esta utilidad, se podrá conectar a cualquier salida de los MessageProcessor del sistema creado. Se podrá registrar diferentes puntos del sistema, con los que podremos volver a comprobar, en caso de ser necesario, los datos que han pasado por el sistema.

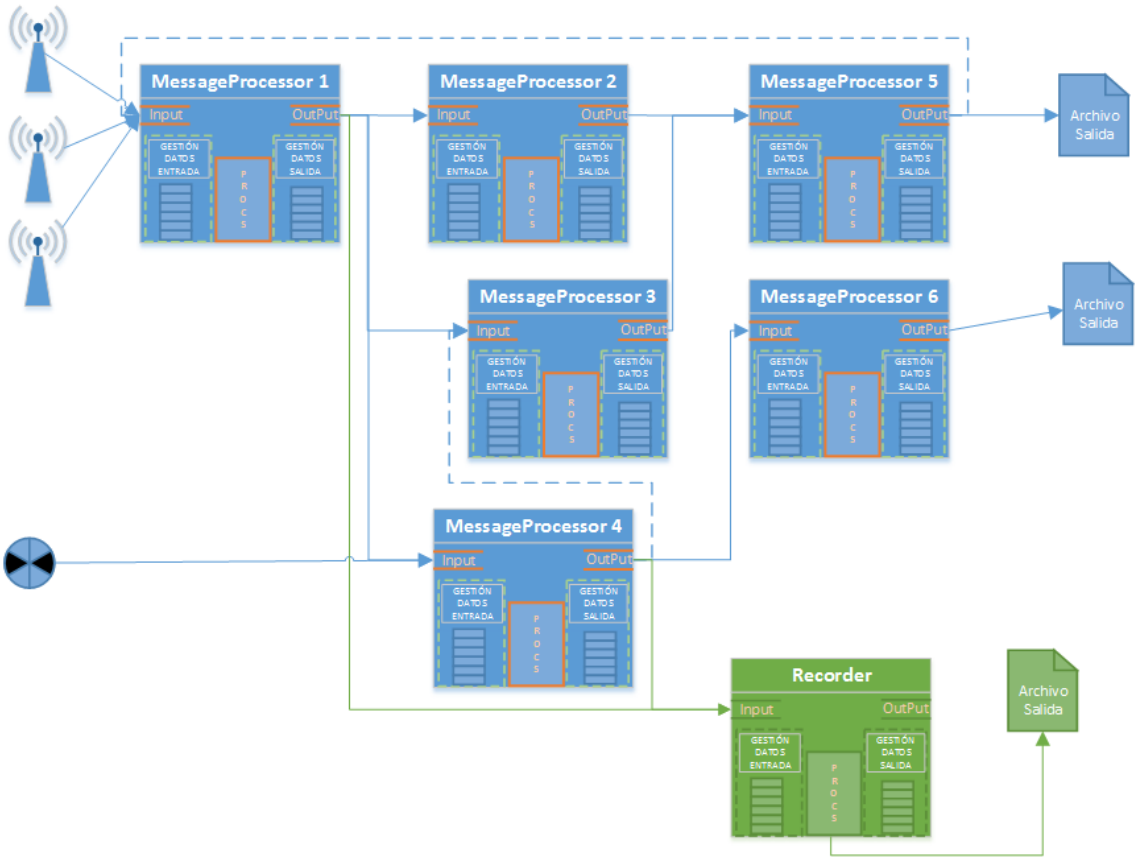


Ilustración 27. Utilidad Recorder

Podemos ver en el ejemplo de la Ilustración 27, como todos los datos recibidos como entrada al sistema, han sido almacenamos en un fichero binario con la utilidad Recorder, esto nos dará la posibilidad de poder volver a reproducirlos con la siguiente utilidad que se explicará a continuación.

4.7.2 Player

Otra de las funcionalidades que ofrece el framework y que permitirá al sistema que lo implemente, es la lectura de los datos generados por MessageRecorder, esta utilidad recibe el nombre player y se lleva a cabo con la clase **MessagePlayer**.

MessagePlayer, es una clase que hereda de MessageProcessor y que incluye la funcionalidad de recuperar los datos desde un fichero que se haya creado previamente con la utilidad Recorder. Con esto conseguiremos volver a reproducir en el sistema, los Mensajes serializados mediante la reconstrucción del mensaje original, y que podrá ser enviado al sistema como si se tratase de la entrada original.

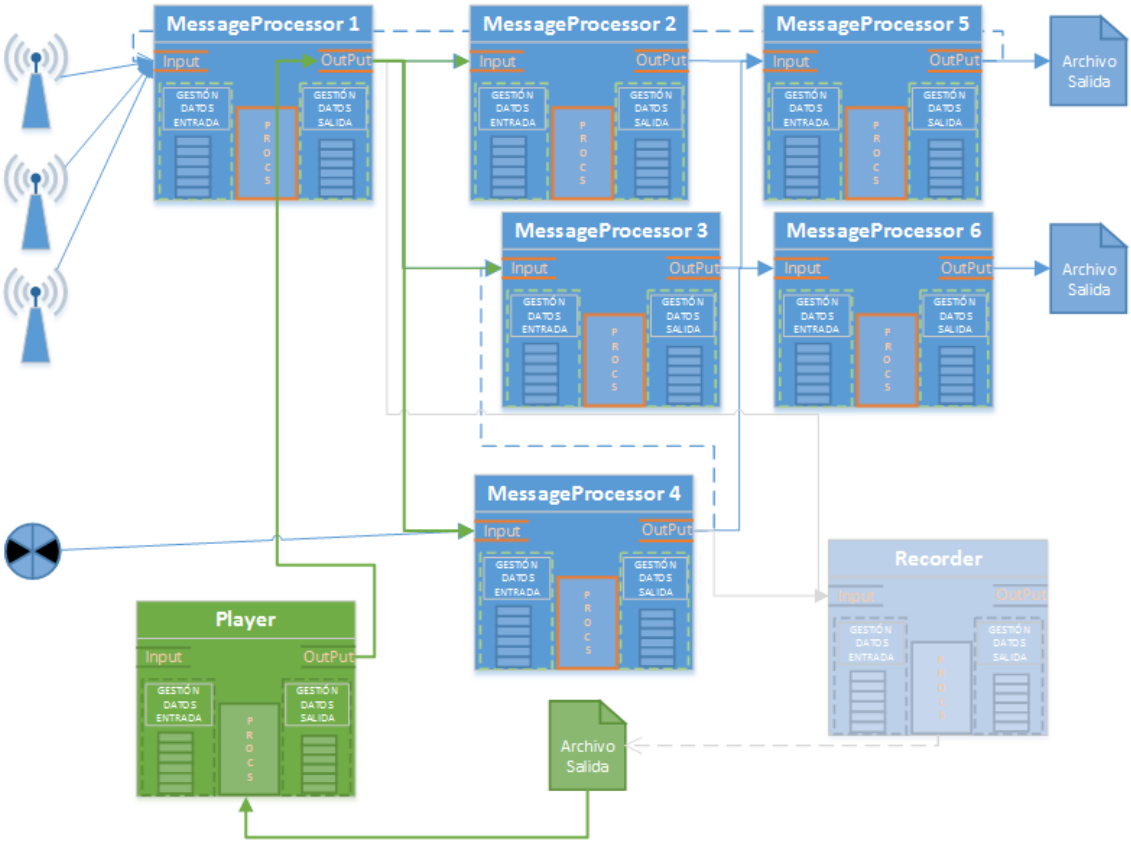


Ilustración 28. Utilidad Player

Como vemos en la Ilustración 28, los datos recuperados de fichero, son emitidos a las entidades que le corresponde, consiguiendo reproducir de esta manera todos los mensajes en las mismas condiciones que en el momento que fueron guardadas.

4.7.3 Sockets

Esta utilidad tiene especial interés, pues nos da la posibilidad de distribuir el procesado con diferentes máquinas que configuremos en el sistema.

Para llevar a cabo esta funcionalidad, será necesaria la configuración de esta utilidad, en los diferentes puntos en los cuales se quieran interconectar.

Ilustración 29. Utilidad Sockets

Como vemos en Ilustración 29, el propósito de esta utilidad, es proporcionar una comunicación entre diferentes máquinas mediante la serialización de los datos en binario, enviándolos a través de la red. Para que esto sea posible, será necesario hacer uso de la utilidad Sockets en cada extremo de la comunicación, que se encargará, en el lado del envió, de serializar estos datos, y en el lado de la recepción, convertir estos datos en forma de mensaje para volver a emitirlos al MessageProcessor que corresponda, simulando de esta manera, una comunicación idéntica a la vista hasta ahora con un pequeño paso de por medio.

4.8 Ejemplos de uso del framework

En este punto se incluirá algunos ejemplos de interconexión y tipos de procesado de los datos donde se podrá ver la funcionalidad del Framework en diferentes situaciones, con el fin de representar y facilitar la comprensión de lo anteriormente detallado.

A lo largo de todas las pruebas la representación de los resultados tendrá el mismo formato. Los datos que se mostrarán en cada prueba serán los siguientes:

- **Entidad**: este campo representa la entidad que muestra o realiza la acción.
- **Value**: este campo corresponde con el valor que tiene el mensaje y será el objeto a procesar en las entidades.
- **Tiempo**: en la unidad de tiempo de milisegundos, se mostrará el momento en el que se ha realizado cada acción.

4.8.1 Ejemplo 1

En este ejemplo tendremos una serie de datos de entrada que se obtendrá desde un fichero y serán leídos a través de la utilidad Player.

Estos datos serán enviados a un MessageProcessor, el cual, se encargará de procesar por tiempo los datos, es decir, este MessageProcessor según recibe los mensajes, los va almacenando en la cola de otro hilo de ejecución y en el momento que llegue la franja de tiempo establecida, procesará en este mismo hilo los datos.

Tras el procesado, el cual consistirá en el incremento en uno del valor recibido, el MessageProcessor se encargará de enviar los resultados, este envío se realizará de forma síncrona, donde el mismo hilo de ejecución que se encarga del procesado y envíar los resultados al siguiente MessageProcessor.

El tercer MessageProcessor, con una configuración síncrona tanto en el envió como en la gestión de los datos de entrada, vemos una especialización de un MessageProcessor que se encarga de mostrar los resultados por pantalla, con esto, podemos separar la parte de procesado, con la de la representación de los datos, pudiendo cambiar el formato de salida con una simple desconexión de esta entidad e

interconectando otra que implemente la funcionalidad para mostrar los datos en otro medio.

El sistema estará formado por las siguientes entidades con la configuración de procesado y de envío especificada en la siguiente tabla.

MessageProcessor	Datos de entrada	Procesado	Transmisión
Entidad 1	Lectura desde fichero, a través de la utilidad Player.	Síncrono	Síncrono
Entidad 2	Datos enviados desde la entidad 1	Por tiempo (3 ms)	Síncrono
Entidad 3	Datos procesados por la entidad 2	Síncrono	Síncrono

Tabla 4. Descripción Ejercicio 1

4.8.1.1 Diagrama

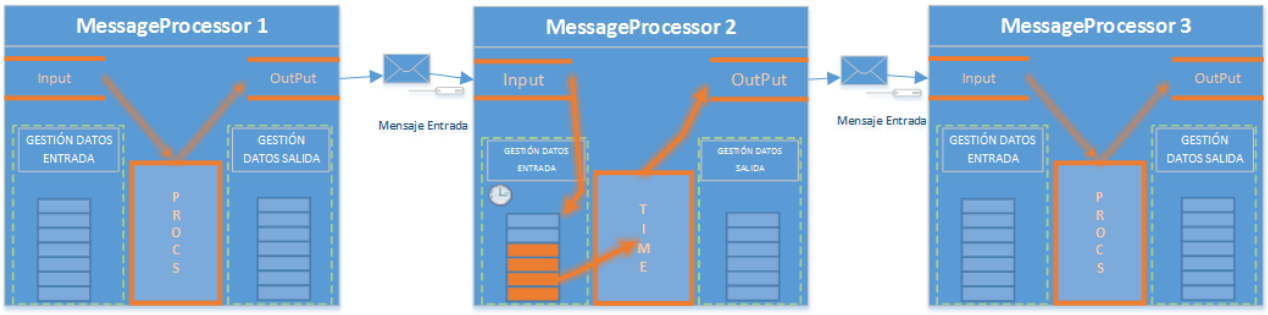


Ilustración 30. Diagrama Ejemplo 1

4.8.1.2 Objetivo

El objetivo será reflejar el funcionamiento del Framework, a través de este ejemplo sencillo, el cual estará formado tres MessageProcessor; uno que se encargará de obtener la información, otro que se encargará del procesado por tiempo de los datos para finalizar con un envío síncrono al MessageProcessor encargado de la salida de los datos por pantalla.

4.8.1.3 Resultados Esperados

En la siguiente tabla, se podrá ver en la primera columna el valor enviados por el MessageProcessor 1, encargado de capturar los datos de fichero y enviar los mensajes al MessageProcessor 2. La otra columna corresponderá con los valores

esperados que debería calcular la entidad 1 y posteriormente enviarlos para ser mostrados por la entidad 3.

Valor Entidad 0	Resultado Entidad1	Valor Entidad 0	Resultado Entidad1
0	1	25	26
1	2	26	27
2	3	27	28
3	4	28	29
4	5	29	30
5	6	30	31
6	7	31	32
7	8	32	33
8	9	33	34
9	10	34	35
10	11	35	36
11	12	36	37
12	13	37	38
13	14	38	39
14	15	39	40
15	16	40	41
16	17	41	42
17	18	42	43
18	19	43	44
19	20	44	45
20	21	45	46
21	22	46	47
22	23	47	48
23	24	48	49
24	25	49	50

Tabla 5. Resultados Esperados Ejemplo 1.

4.8.1.4 Resultado del ejemplo

Los siguientes resultados, corresponden con la salida por pantalla generada por la entidad 3, la cual especificará de donde ha recibido el mensaje, el valor que tiene dicho mensaje como resultado del procesado y el tiempo en el cual se ha procesado y generado el mensaje.

Entidad	Valor	Tiempo	Entidad	Valor	Tiempo
2	1	1370433525927	2	26	1370433525927
2	2	1370433525927	2	27	1370433525927
2	3	1370433525927	2	28	1370433525927
2	4	1370433525927	2	29	1370433525927
2	5	1370433525927	2	30	1370433525927
2	6	1370433525927	2	31	1370433525927
2	7	1370433525927	2	32	1370433525927
2	8	1370433525927	2	33	1370433525927
2	9	1370433525927	2	34	1370433525928
2	10	1370433525927	2	35	1370433525928
2	11	1370433525927	2	36	1370433525928
2	12	1370433525927	2	37	1370433525928
2	13	1370433525927	2	38	1370433525928
2	14	1370433525927	2	39	1370433525928
2	15	1370433525927	2	40	1370433525928
2	16	1370433525927	2	41	1370433525928
2	17	1370433525927	2	42	1370433525931
2	18	1370433525927	2	43	1370433525931
2	19	1370433525927	2	44	1370433525931
2	20	1370433525927	2	45	1370433525931
2	21	1370433525927	2	46	1370433525931
2	22	1370433525927	2	47	1370433525931
2	23	1370433525927	2	48	1370433525931
2	24	1370433525927	2	49	1370433525931
2	25	1370433525927	2	50	1370433525931
Número de mensajes recibidos de la entidad 1= 50					

Tabla 6. Resultados Ejemplo 1

4.8.1.5 Conclusión de los resultados

Tras los resultados obtenidos, se puede observar como la configuración fijada para un procesamiento por tiempos cada 3 milisegundos, ha funcionado, por ello se ha generado un número de paquetes lo suficiente elevado para que se produzca 2 ciclos de ejecución sobre el MessageProcessor 2.

En el primer bloque de procesamiento nos encontramos los 41 primeros mensajes, los cuales han sido procesados en el instante de tiempo 27 y 28. El segundo bloque, vemos como pasados 3 milisegundos se comienza a procesar en el instante de tiempo 31 donde se realiza el resto del procesado de los mensajes que no habían llegado en el primero bloque, lo que corresponde con los mensajes del 42 al 50.

En cuanto a los resultados obtenidos podemos ver que corresponden con los esperados en la tabla superior, donde el MessageProcessor 2 ha ido incrementando en uno el valor que ha recibido.

4.8.2 Ejemplo 2

El ejemplo dos es una pequeña ampliación del primero. A este ejemplo se le ha añadido un MessageProcessor más con el que podremos reflejar sobre los mismos datos de entrada, un procesado distinto al mismo tiempo. Este nuevo MessageProcessor se conectará a la salida del MessageProcessor 1 y será el encargado de realizar el procesado al mismo tiempo que el MessageProcessor 2.

MessageProcessor	Datos de entrada	Procesado	Transmisión
Entidad 1	Lectura desde fichero, a través de la utilidad Player.	Síncrono	Síncrono
Entidad 2	Datos enviados de la entidad 1	Por tiempo(1 ms)	Síncrono
Entidad 3	Datos enviados por la entidad 1	Asíncrono	Por tiempo(2 ms)
Entidad 4	Datos procesados por la entidad 2 y 3.	Síncrono	Síncrono

Tabla 7. Descripción Ejercicio 2

4.8.2.1 Diagrama

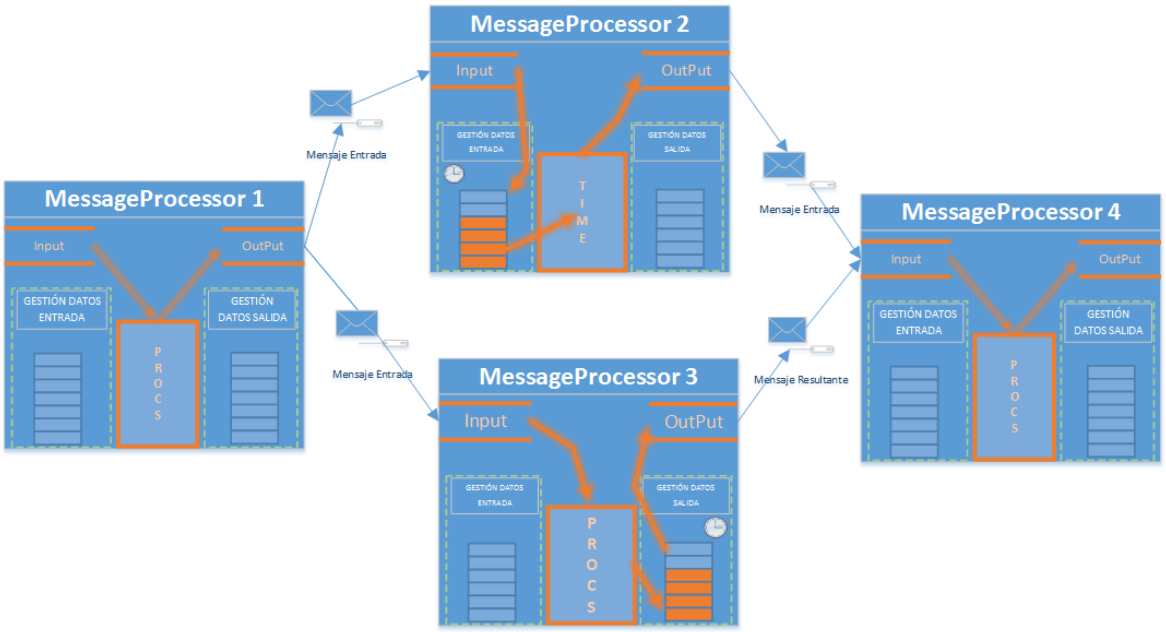


Ilustración 31. Diagrama Ejemplo 2

4.8.2.2 Objetivo

El objetivo del ejemplo es demostrar las posibilidades que permite el framework para realizar diferentes tipos de procesado sobre un mismo conjunto de datos.

En el MessageProcessor 2 se llevará a cabo el mismo procesado sobre los datos, por lo que se incrementará en uno los datos recibidos y en el MessageProcessor 3 realizaremos otro procesado en los datos que consistirá en el decremento en uno de los datos recibidos.

Se pretende demostrar la posibilidad de realizar un procesado de los datos diferente en cada MessageProcessor con el fin de ajustarse a las necesidades del problema. Por ejemplo en el MessageProcessor 2 puede ser de interés realizar un procesado de los datos sobre un bloque de datos, los cuales tras una franja de tiempo establecida pasará a ser enviados. Sin embargo en el MessageProcessor 3 es posible que sea más conveniente el tratamiento de los datos en el momento de la recepción para no acumular una carga de trabajo elevada y realizar el envío de los datos procesados en conjunto tras una franja de tiempo estipulada.

4.8.2.3 Resultados Esperados

En la siguiente tabla se podrá ver los valores enviados por el MessageProcessor 1, encargada de capturar y enviar los mensajes a los MessageProcessor 2 y 3. Las otras dos columnas corresponderán con los valores esperados que debería calcular cada uno de los MessageProcessor y posteriormente mostrarnos el MessageProcessor 4.

Valor Entidad 1	Resultado Entidad 2	Resultado Entidad 3
0	1	-1
1	2	0
2	3	1
3	4	2
4	5	3
5	6	4
6	7	5
7	8	6
8	9	7
9	10	8

Tabla 8. Resultados Esperados Ejemplo 2

4.8.2.4 Resultado del ejemplo

Los siguientes resultados corresponden con la salida por pantalla generada por el MessageProcessor 4, el cual especificará de donde ha recibido el mensaje, el valor que tiene dicho mensaje y el tiempo en el cual se ha procesado y generado el mensaje.

Entidad	Valor	Tiempo
2	1	1370376599545
2	2	1370376599545
2	3	1370376599545
2	4	1370376599545
2	5	1370376599545
2	6	1370376599545
2	7	1370376599545
2	8	1370376599545
2	9	1370376599545
2	10	1370376599545
3	-1	1370376599544
3	0	1370376599544
3	1	1370376599544
3	2	1370376599544
3	3	1370376599544
3	4	1370376599544
3	5	1370376599544
3	6	1370376599544
3	7	1370376599544
3	8	1370376599544
Número de mensajes recibidos de la entidad 2= 10		
Número de mensajes recibidos de la entidad 3= 10		

Tabla 9. Resultados Ejemplo 2

4.8.2.5 Conclusión de los resultados

En la salida por pantalla que proporciona el MessageProcessor 4 podemos comprobar los resultados, los cuales, corresponden con los esperados.

Se muestran los resultados por entidad para comprobar los valores tras su procesado y comprobar si han realizado las operaciones correctamente.

Algo en lo que nos podemos fijar es en los tiempos de los mensajes, este tiempo es el momento en el que se procesa los datos y se genera dicho mensaje. Por lo tanto, al comprobar la configuración que tenemos en nuestro sistema, vemos que los mensajes han sido procesados antes en el MessageProcessor 3, esto se debe a la configuración síncrona, lo que indica que se debe procesar según van llegando los mensajes. Sin embargo, en el MessageProcessor 2 se han procesado un poco más tarde por la configuración de procesado que tenemos establecida, la cual está fijado por tiempo, lo que nos lleva a tener que esperar el milisegundo que tiene como franja de tiempo en la configuración para ser procesados los datos.

Por lo tanto estamos viendo que los datos aunque han sido emitidos en el mismo momento, se han procesado en distintos tiempos por la configuración de procesado establecida.

Si vamos un poco más en detalle sobre los resultados que han salido por pantalla, podemos ver que los datos que primero se han mostrado, son los del MessageProcessor 2. ¿Por qué?, bueno si nos fijamos en el tiempo que tiene la gestión de los datos de envío en el MessageProcessor 3, podemos ver que este tiempo son 2 milisegundos. Lo que provoca que a pesar de tener procesado los mensajes, se debe esperar 2 milisegundos para realizar el envío. Por ello, el MessageProcessor 2 al tener un envío de los datos síncrona, envía los mensajes antes y podemos ver en la pantalla que estos son mostrados antes.

4.8.3 Ejemplo 3

El ejemplo 3 será una extensión del ejemplo 2 para facilitar la compresión y poder ver poco a poco como se puede ir incrementando la complejidad del sistema.

En este caso seguirán existiendo las entidades 2 y 3 que realizarán el mismo procesado de los datos que en el ejemplo anterior, pero en esta ocasión, el procesado lo realizarán con la misma configuración, por eventos. Este evento será recibido en el mismo instante por las dos entidades que realizarán el procesado simultáneamente y se enviarán los mensajes procesados por bloques tras una franja de tiempo fija, donde ambas entidades tendrán el mismo tiempo establecido.

El nuevo MessageProcessor que se ha incluido recibirá el conjunto de mensajes procesados por ambas entidades y se encargará de multiplicar los resultados de los diferentes mensajes, multiplicando el mismo número de mensaje de ambos MessageProcessor, consiguiendo de esta manera la multiplicación del incremento y decremento del mismo dato.

Por ejemplo, para facilitar la comprensión del objetivo, si el dato enviado por el MessageProcessor 1 es 50, en el MessageProcessor 2 procesará el dato y sacará como resultado 51, y en el MessageProcessor 3 como resultado del procesado enviará 49, por lo tanto, en el MessageProcessor 4 se encargará de multiplicar (51 * 49).

Como elemento de salida, el MessageProcessor 4 se encargar de enviar el resultado a los MessageProcessor 5 y 6 que tendrán como objetivo saca el resultado en un fichero y mostrará el resultado por pantalla.

MessageProcessor	Datos de entrada	Procesado	Transmisión
Entidad 1	Lectura desde fichero, a través de la utilidad Player.	Síncrono	Síncrono
Entidad 2	Datos enviados por la entidad 1	Por evento	Por tiempo
Entidad 3	Datos enviados por la entidad 1	Por evento	Por tiempo
Entidad 4	Datos procesados por la entidad 2 y 3.	Asíncrono	Síncrono
Entidad 5	Datos recibidos por la entidad 4.	Síncrono	Síncrono
Entidad 6	Datos recibidos por la entidad 4	Síncrono	Síncrono

Tabla 10. Descripción Ejercicio 3

4.8.3.1 Diagrama

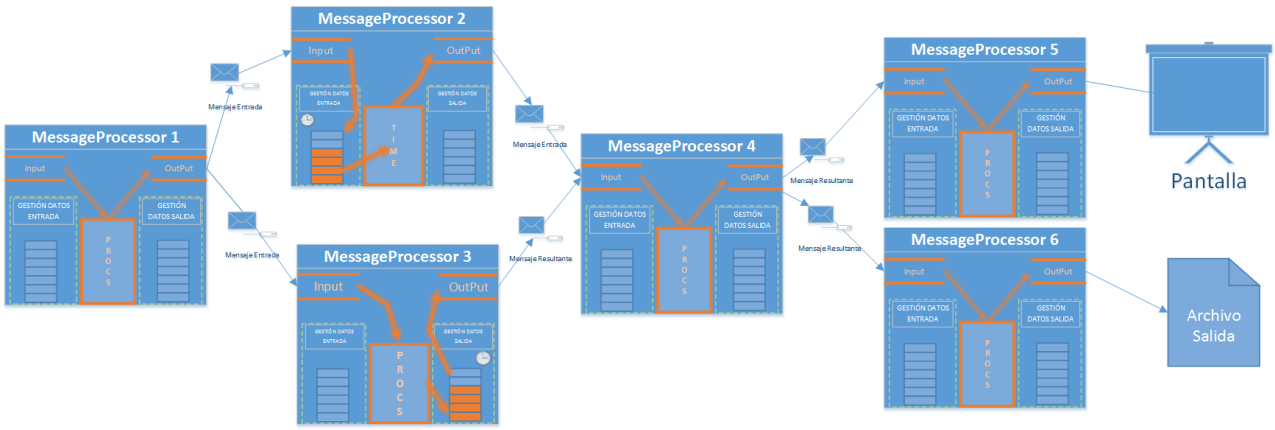


Ilustración 32. Diagrama Ejemplo 3

A través del diagrama facilitado en la Ilustración 32, podemos ver el problema planteado y como se resuelve las intercomunicaciones del sistema creado.

4.8.3.2 Objetivo

El propósito de este ejemplo, es mostrar la sincronización de diferentes MessageProcessor para enviar a un MessageProcessor en común y procesar los datos de ambos.

Otro aspecto que se quiere reflejar en el ejemplo, es la posibilidad de configurar la salida de un dato a varias salidas y como esto se puede realizar en cualquier momento añadiendo un MessageProcessor que se conecte a la salida de otro MessageProcessor y tratarlo independientemente.

4.8.3.3 Resultados Esperados

En la siguiente tabla se podrá ver los valores enviados por la entidad 0, encargada de generar y enviar los mensajes a las entidad 1 y 2. Las otras dos columnas corresponderán con los valores esperados que debería calcular cada una de las entidades y posteriormente la entidad 3 calculará otro resultado de la entrada de ambas entidades y lo enviará a entidad 5 para ser mostrado por pantalla.

Valor Entidad 1	Resultado Entidad 2	Resultado Entidad 3	Resultado Entidad 4
0	1	-1	- 1
1	2	0	0
2	3	1	3
3	4	2	8
4	5	3	15
5	6	4	24
6	7	5	35
7	8	6	42
8	9	7	63
9	10	8	80

Tabla 11. Resultados Esperados Ejemplo 3

4.8.3.4 *Resultado del ejemplo*

Los siguientes resultados corresponden con la salida por pantalla generada por la entidad 3, la cual especificará de donde ha recibido el mensaje, el valor que tiene dicho mensaje y el tiempo en el cual se ha procesado y generado el mensaje.

Salida por pantalla			Salida de fichero		
Entidad	**Valor**	**Tiempo**			
4	-1	1370436518987	4	-1	1370436518987
4	0	1370436518987	4	0	1370436518987
4	3	1370436518987	4	3	1370436518987
4	8	1370436518987	4	8	1370436518987
4	15	1370436518987	4	15	1370436518987
4	24	1370436518987	4	24	1370436518987
4	35	1370436518987	4	35	1370436518987
4	48	1370436518987	4	48	1370436518987
4	63	1370436518987	4	63	1370436518987
4	80	1370436518987	4	80	1370436518987
Número de mensajes recibidos de la entidad 4 = 10			Número de mensajes recibidos de la entidad 4=10		

Tabla 12. Resultados Ejemplo 3

4.8.3.5 *Conclusión de los resultados*

En primer lugar vemos que los resultados tanto por pantalla como por fichero corresponden con los esperados, por lo tanto concluimos que las operaciones se han realizado correctamente entre todas las entidades.

Al ser un procesado de un número tan reducido de datos vemos que se han realizado en el transcurso del milisegundo 87.

Por último, destacar que la salida tanto por pantalla como la de fichero contienen los mismos datos ya que las entidades 5 y 6 solo se encargan de sacar los datos recibidos por la entidad 4 y no realizar ninguna transformación de los datos, simplemente son una especialización que se encargan de mostrar el resultado a diferentes medios.

APLICACIÓN DEL FRAMEWORK EN EL ENTORNO DE LA FUSIÓN DE LOS DATOS

5 Aplicación del Framework en el entorno de la fusión de los datos.

En este punto, se llevará a la práctica sobre un problema real el framework desarrollado. Como se ha detallado en la motivación, este fue el principal objetivo por el cual se vio la necesidad de crear un framework genérico capaz de realizar un procesado de los datos.

Con la ayuda de este sistema se conseguirá reflejar y demostrar, las posibilidades, las utilidades y el correcto funcionamiento del framework creado. Con la aplicación del framework sobre el entorno de la fusión de los datos, conseguiremos someterle sobre diferentes tipos de situaciones, que lo elevarán a su máxima explotación con una gran carga de trabajo, consiguiendo llegar a forzar al framework a sus máximas posibilidades.

5.1 Introducción

El problema consiste en el tratamiento de los datos recibidos por dos tipos de sensores: un conjunto de radares y una serie de estaciones de seguimiento AIS (Automatic Identification System), que permiten a los buques comunicar su posición y otros datos relevantes sobre su situación y características. Estos dos tipos de sensores proporcionan datos complementarios que se pueden fusionar para tener una mejor información de lo que sucede en el espacio marítimo y costero de la zona de interés.

Con el software resultante lo que se permite es realizar una mejor vigilancia marítima, integrando de forma simultánea las capacidades de los radares y las estaciones de localización AIS desplegados. El objetivo es garantizar la seguridad de la zona controlando las diversas naves que circulan por una determinada vía marítima que, a su vez, es entrada y salida de un puerto comercial.

Para llevarlo a cabo, se requiere el procesado de los datos, consiguiendo como elemento final una imagen completa, precisa y actualizada, similar a la que se ofrece a los controladores aéreos, de todas las naves que se encuentran en la zona de cobertura para poder gestionar adecuadamente el tráfico marítimo y detectar anomalías con la mayor anticipación posible.

Será en la fusión de los datos recibidos por los sensores donde más carga de trabajo tenga el procesado de los datos y donde se proporcione un mayor paralelismo en el sistema para ofrecer el máximo rendimiento.

A modo de ejemplo, en la Ilustración 33 se puede representar de forma esquemática la situación del problema, donde observar los diferentes elementos que componen el sistema, su función y el resultado de la detección de un posible barco en un supuesto escenario.

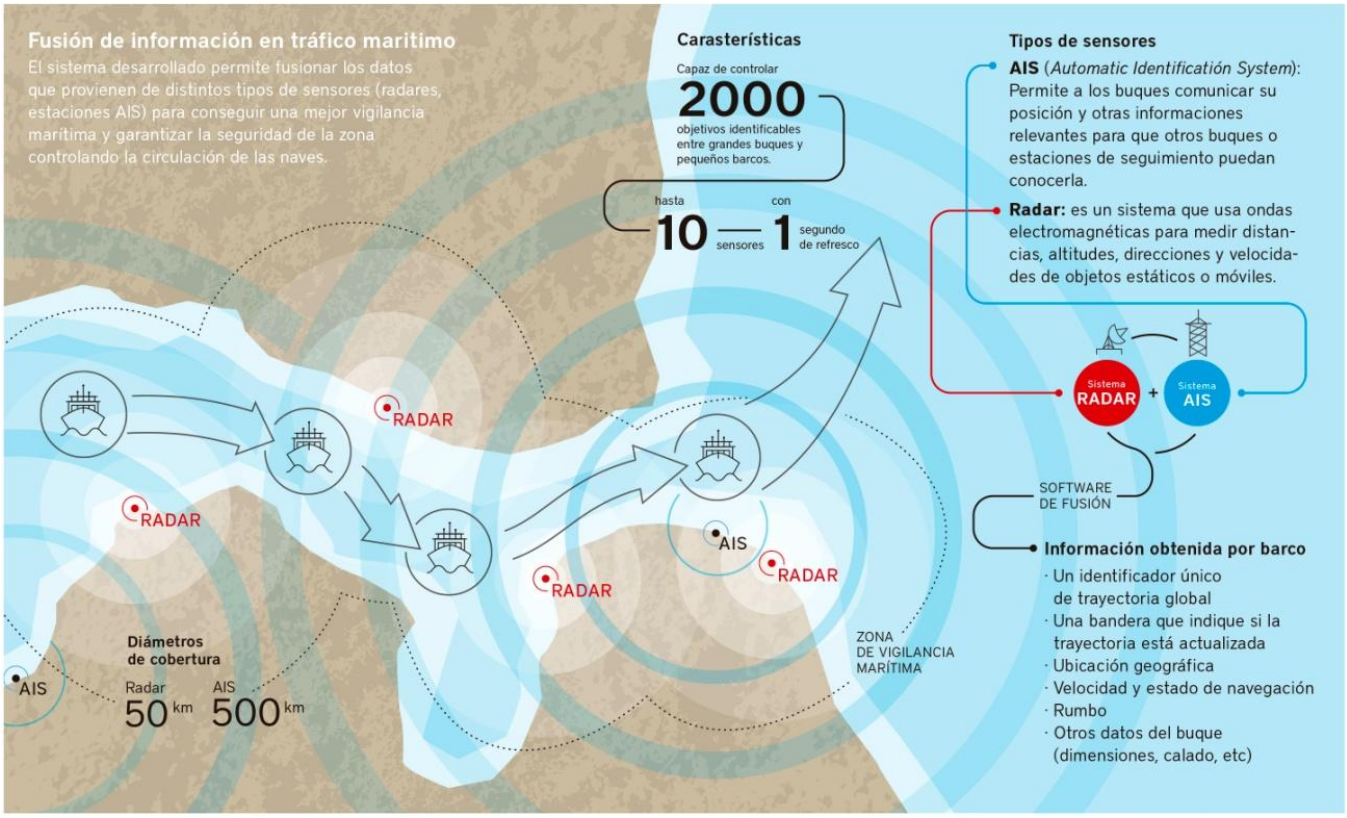


Ilustración 33. Representación de un escenario en el Framework de Fusión

Como se puede observar en la Ilustración 33, a través de las detecciones de los radares y de la información proporcionada por el barco a la estación AIS, el sistema debe ser capaz de representar la ruta que lleva el barco con la fusión de los datos obtenidos de los diferentes sensores que se encuentran en el escenario. El sistema, nos ayudará a predecir la siguiente posición con un pequeño margen de error, basándonos en los datos obtenidos a lo largo de la trayectoria, datos como la velocidad, la dirección en la que se dirige y el posible giro que puede realizar a la velocidad que se encuentra, permitiendo evitar así, posibles colisiones entre todos los elementos que se encuentren en el escenario.

5.2 Análisis del sistema

Como se ha mencionado en la introducción, el sistema tiene como objetivo final mostrar la fusión de las detecciones recibidas por el conjunto de sensores que forman el escenario. Para que esto sea posible, el sistema debe ser capaz de procesar la información recibida, lo que supondrá realizar una serie de funciones.

Ilustración 34 . Funciones Framework de la fusión de los datos

Recepción y Transformación de la señal

En primera instancia como se puede ver en la Ilustración 34, nos encontramos con la recepción de las señales de los sensores. El problema con el que nos encontramos al recibir estos datos es que procesar la información en ese formato supone un incremento muy elevado en el procesamiento, por ello, será necesario hacer una transformación de los datos a coordenadas cartesianas.

Asociación de las señales por sensor

Una vez transformadas las señales, nos encontraremos en disposición de procesar estos datos de una forma más eficiente. Con el propósito de ofrecer un paralelismo sobre la carga de trabajo de los datos recibidos, se procesarán las señales de cada sensor por separado, asociando las detecciones recibidas a posibles trayectorias existentes en el sistema o por el contrario creando nuevas trayectorias. Esta acción nos llevará a la tercera función a desarrollar en el sistema, la cual, tendrá como datos de salida todas las trayectorias que se han detectado en cada sensor.

Asociación de las trayectorias de todos los sensores

Por último, tras la asociación de los datos por sensor, nos centraremos en una función común a todos los sensores, la fusión de todas las trayectorias detectadas en cada uno de los sensores.

Como se ha podido ver en la introducción, en el entorno nos encontraremos con más de un sensor, lo que supondrá tener en el sistema implementado cada uno de ellos. Por lo tanto, el sistema se puede representar de la siguiente forma.

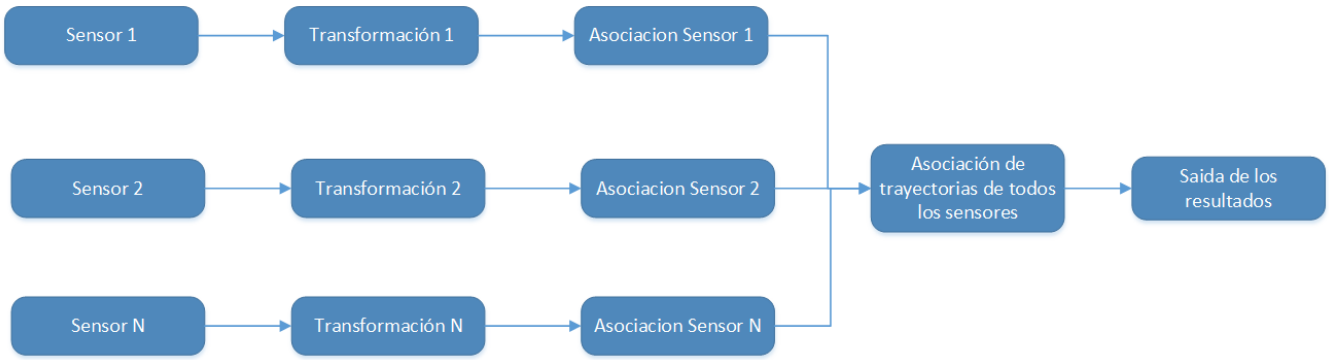


Ilustración 35 . Funciones Framework de la fusión de los datos 2

En la Ilustración 35, se pueden ver las funciones que deben resolver el sistema y las partes que se realizarán en paralelo. A estas funciones, se las podrán aplicar un paralelismo realmente eficiente, gracias a la no dependencia de los datos con la que nos encontramos. Cada sensor, dispondrá de sus datos de forma independiente sin la compartición de los mismos. Será en la parte de la asociación de las trayectorias donde se unifiquen estos datos y se saque el resultado en común.

5.3 Diseño de los componentes

Una vez detectadas las funciones a desarrollar en el sistema, pasaremos a la creación de las mismas.

Para poder llevar a cabo el sistema sobre nuestro framework, cada función que requiere el sistema, se llevará a cabo sobre un MessageProcessor.

En el siguiente diagrama, podemos ver la fácil adaptación que se ha utilizado como solución final al problema del sistema a implementar.

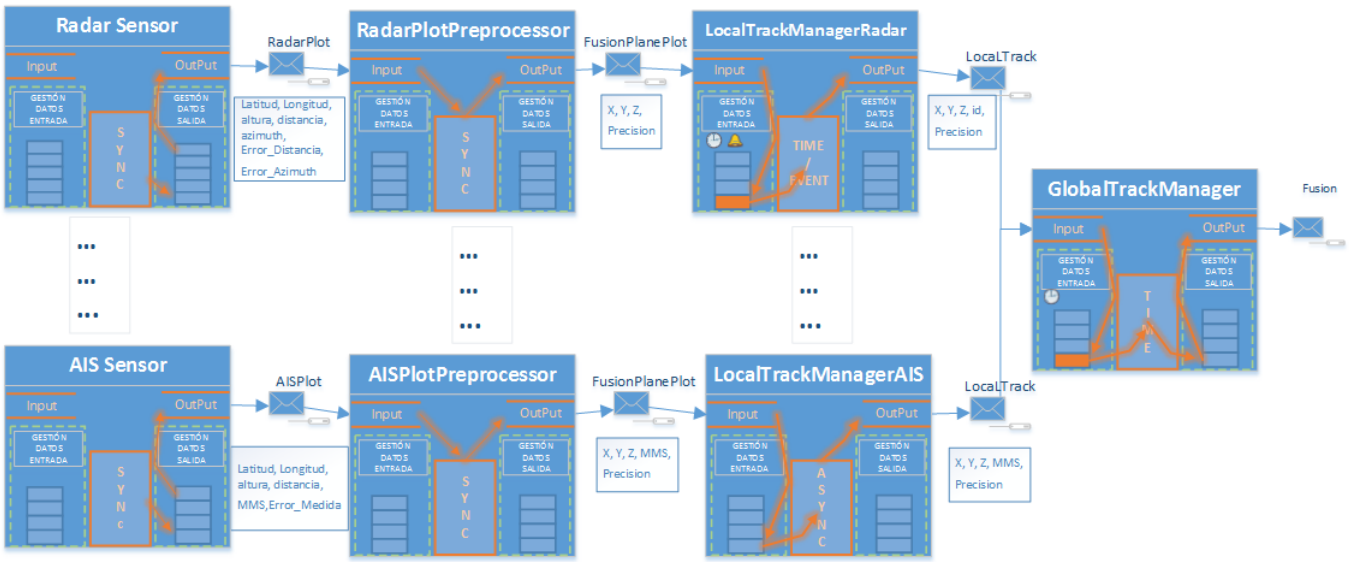
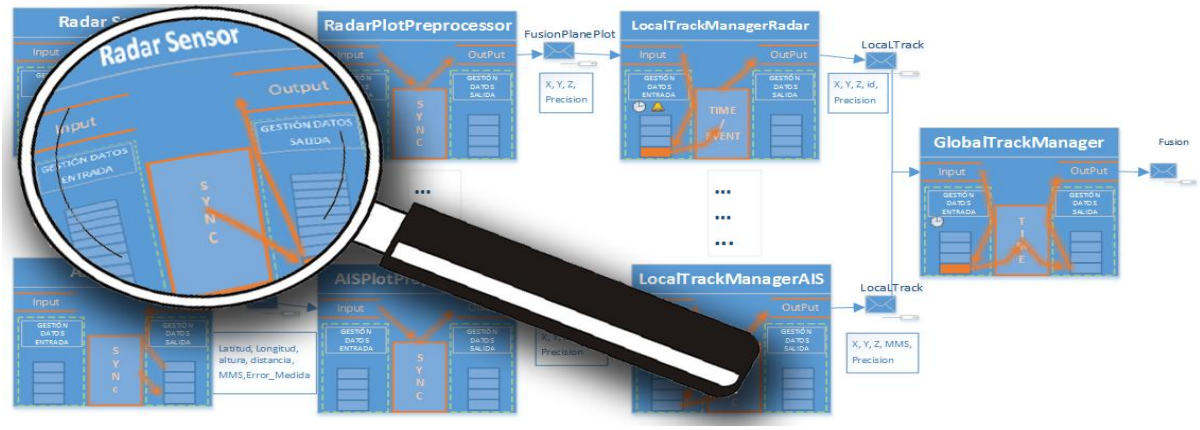


Ilustración 36. Sistema de la fusión de los datos

En la Ilustración 36. Sistema de la fusión de los datos, hay que mencionar que es un diagrama que pretende reflejar cada uno de los componentes existentes que forma el sistema y sus interconexiones, pero no refleja el sistema al completo, donde existiría una cadena por cada tipo de sensor existente en el sistema, e incluso, como se podrá ver en futuras secciones, la conexión de las diferentes utilidades que se proporcionan en el framework.

A continuación, se explicará cada uno de los elementos con detalle para reflejar las decisiones adoptadas en cada una de las entidades desarrolladas y el objetivo y función que cumplen en el sistema.

5.3.1 Sensores



Como primer elemento a destacar del sistema se detallará los sensores. Uno de los elementos más importantes del sistema, los cuales, se encargan de proporcionar toda la información a procesar.

En el sistema contaremos con dos tipos de sensores, los radares y los AIS. Es importante conocer su funcionamiento para entender el tipo de información que nos proporcionará cada uno de ellos.

La gran diferencia entre ellos, está en que el radar, detecta dentro de su alcance los elementos mediante el rebote de las ondas electromagnéticas que él mismo emite de una forma aproximadamente periódica según el periodo de rotación de la antena. Sin embargo, a los sensores AIS les proporcionan medidas de posición en coordenadas globales esféricas (WGS-84[1]), velocidad y código único identificativo del buque. Los AIS son una fuente periódica, según la frecuencia de envío de mensajes del transpondedor, que depende de la velocidad del barco, por lo que es posible darse con algunas irregularidades.

Independientemente del tipo de sensores, en el sistema como elemento final los dos sensores proporcionarán el mismo tipo de datos, estos datos se transmitirán en forma de mensaje con algunas diferencias en los atributos que contiene. A este mensaje nos referiremos a lo largo del documento como **Plots.**

5.3.1.1 Radar Sensor

En el caso de ser generados por los sensores de tipo radar, el MessageProcessor recibirá el nombre de RadarSensor y los mensajes generados por este

[1] World Geodetic System

MessageProcessor, serán llamados **RadarPlots**. El contenido de los RadarPlots tendrán la siguiente información.

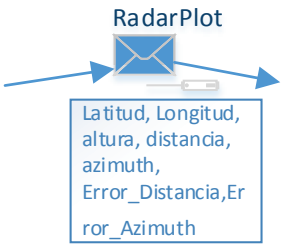


Ilustración 37. Mensaje Tipo RadarPlot

La medida de posición será proporcionada en coordenadas polares (r, θ) y la distancia con altura corregida por la proyección sobre el plano.

Atributos	Definición
Latitud	Latitud de la pista en WGS84 (medida en grados)
Longitud	Longitud de la pista en WGS84 (medida en grados)
Distancia	Distancia (medida en metros)
Azimut	Ángulo (medido en grados)
Error_Distancia	Posible error en la distancia recibida
Error_Azimuth	Posible error en la Azimut recibido

Tabla 13. Atributos Mensaje RadarPlot

Si observamos la Ilustración 38, podemos ver exactamente cómo es la configuración que se le ha dado a esta entidad.

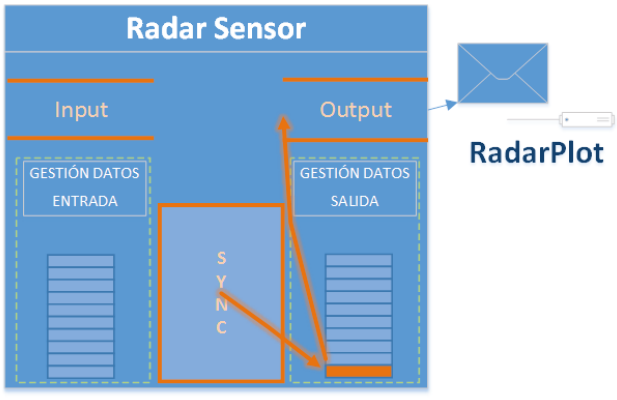


Ilustración 38. MessageProcessor Radar Sensor

Para empezar vemos como la entidad no recibe mediante mensajes para los datos de entrada, esto se puede apreciar por no tener un mensaje de entrada en la fase de Input. Esta entidad, recibirá los datos a través de una comunicación por sockets. Por lo tanto, será parte del procesado la captura de los datos recibidos por los sensores.

Ya que el procesamiento que requiere esta entidad no supone una carga de trabajo elevada, se le ha establecido un procesamiento síncrono, en el cual, será capaz de realizar el procesamiento de estos datos según se van recibiendo. Sin embargo, para garantizar que no se encontrará con ninguna entidad en la cadena del paso de mensajes que pueda provocarle un bloqueo, se ha diseñado una gestión de los datos de salida de forma asíncrona, delegando el envío de los datos en otro hilo de ejecución donde en el caso de encontrarse con un bloqueo inesperado, será capaz de almacenar en su cola de mensajes, los mensajes hasta poder realizar el envío, mientras tanto el MessageProcessor Radar Sensor puede seguir procesando los datos sin repercutirle los problemas encontrados.

5.3.1.2 AIS Sensor

En el caso de ser los sensores de tipo AIS los que generarán los mensajes recibirán el nombre de **AISPlot** y el MessageProcessor tendrá el nombre AIS Sensor. Los mensajes emitidos por este MessageProcessor contendrán la siguiente información.

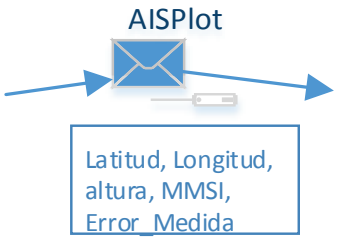


Ilustración 39. Mensaje AISPLot

En la siguiente Tabla 14, se espec2ificará cada uno de los atributos del mensaje.

Atributos	Definición
Latitud	Latitud de la pista en WGS84 (medida en grados)
Longitud	Longitud de la pista en WGS84 (medida en grados)
Altura	Información de la altura a la que se encuentra el Barco(Medida en metros, siempre 0)
MMSI	Identificador proporcionado por el Blanco
Error_Medida	Margen de error en las medidas proporcionadas.

Tabla 14. Atributos Mensaje AISPlot

Como se puede observar en los atributos que proporciona el radar AIS, los atributos son distintos pero el campo de más importancia que se debe destacar es el **MMSI**, este campo corresponde con el identificador del barco. A simple vista puede parecer que no es algo relevante, pero si nos centramos en el procesado que se realiza en

este sistema, la función principal es identificar y relacionar las detecciones que los sensores detectan, por lo que en el caso del AIS esta función se simplifica al recibir en el mensaje, el identificador del barco y pudiéndolo relacionar perfectamente.

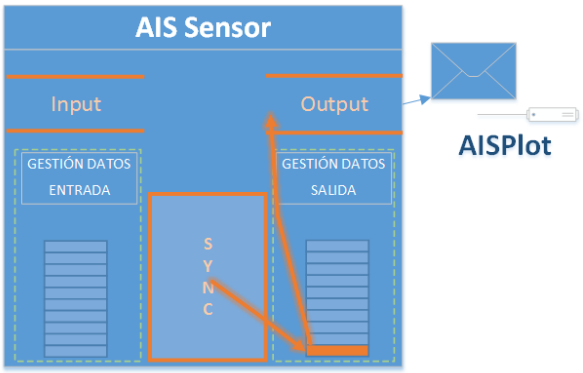


Ilustración 40. MessageProcessor AISSensor

En la Ilustración 40. MessageProcessor AISSensor, podemos ver la configuración asignada al MessageProcessor encargado de recibir las señales de los sensores de tipo AIS.

En primer lugar, al igual que en MessageProcessor Radar Sensor, podemos destacar que las señales no llegan en modo de mensajes puesto que es una entrada desde el exterior al sistema y esta comunicación se realiza a través de una comunicación por sockets. Por consiguiente, se puede decir que la entrada de los mensajes forma parte de la fase de procesado del MessageProcessor.

En este caso al igual que en Radar Sensor, el procesado no consiste en otra cosa más que recibir las señales y transmitirlas a los MessageProcessor asociados a este, permitiendo asignarle una configuración síncrona en el procesamiento de los datos.

En la parte del envío de los datos, usaremos la misma lógica que en el Radar Sensor donde se le aplica una gestión de los datos de salida de forma asíncrona, la cual nos permita delegar la función a otro hilo de ejecución y sea este, el que sufra las consecuencias de cualquier anomalía en el sistema, dejando a un lado a la recepción de los datos para que no le repercuta y pueda seguir el transcurso del procesado sin inferir los problemas ocurridos.

5.3.2 Preprocesado de la información

El siguiente elemento con el que nos encontraremos, será **RadarPlotPreProcessor** y **AISPlotPreProcessor**.

En este MessageProcessor se encarga de convertir las posiciones recibidas por los sensores a medidas cartesianas. Esta conversión es necesaria debido a la gran diferencia en el rendimiento a la hora de trabajar con los datos. Por lo tanto, los MessageProcessor RadarPlotPreProcessor y AISPlotPreProcessor serán los que reciban los mensajes RadarPlot y AISPlot generados por los MessageProcessor Radar Sensor y AIS Sensor.

Es importante tener en cuenta la diferencia entre las fuentes de información recibidas. El sistema AIS proporciona la posición global en coordenadas geodésicas globales (latitud, longitud y altura sobre el elipsoide WGS-84), mientras que el radar proporciona medidas polares locales (distancia, azimut) con respecto a su localización absoluta. También se dispone de información de velocidad en el caso del AIS, expresada en el plano local a la posición del objeto, por tanto distinta al sistema de referencias centrado en el sensor. Es por ello por lo que se precisa de funciones de transformación de coordenadas para expresar todas las medidas con respecto a una referencia común antes de su fusión y así mismo para trasladar el resultado a las coordenadas de salida requeridas.

Como resultado del procesamiento de los dos MessageProcessor, se puede observar que el mensaje que generan, es del mismo tipo tras las correspondientes transformaciones, este mensaje, recibirá el nombre de **FusionPlanePlot.** Los campos de este mensaje tiene los siguientes valores.

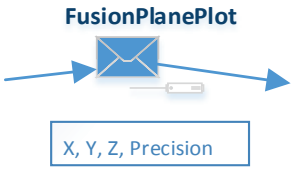


Ilustración 41. Mensaje FusionPlanePlot

Atributos	Definición
x	Posición sobre el eje de coordenadas X
y	Posición sobre el eje de coordenadas Y
z	Posición sobre el eje de coordenadas Z
Precision	Precisión calculada de la medida

Tabla 15. Atributos FusionPlanePlot

Se puede ver que tras la conversión de las señales, el resultado del mensajes se convierte en un conjunto de atributos fáciles de manejar, serán 3 campos que representará la posición sobre el eje de coordenadas y una precisión sobre el cálculo de esta posición.

5.3.2.1 *RadarPlotPreProcessor*

Como se ha detallado, este MessageProcessor se encarga de transformar las señales recibidas a medidas cartesianas, este proceso no requiere un procesado elevado.

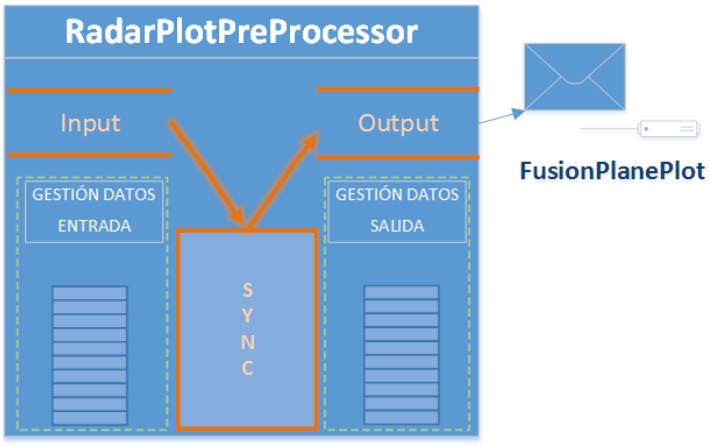


Ilustración 42. MessageProcessor RadarPlotPreProcessor

Como se puede ver en la Ilustración 42. MessageProcessor RadarPlotPreProcessor, debido a la velocidad en la transformación de una media a la otra, y teniendo en cuenta que el envío por los MessageProcessor anteriores han sido realizados de forma asíncrona, le podremos asignar una configuración en la gestión de los datos de

entrada y en la gestión de los datos de salida de forma síncrona, lo que supondrá que el mismo hilo de ejecución se encargue de realizar toda la gestión de los datos.

5.3.2.2 AISPlotPreProcessor

Como hemos podido ver en Ilustración 36. Sistema de la fusión de los datos, para el caso de las señales recibidas por sensores AIS nos encontraremos con el MessageProcessor AISPlotPreProcessor.

Este MessageProcessor será el encargado de transformas las señales recibidas de coordenadas geodésicas globales, a medidas cartesianas, con el fin de poder trabajar conjuntamente con los sensores de tipo Radar.

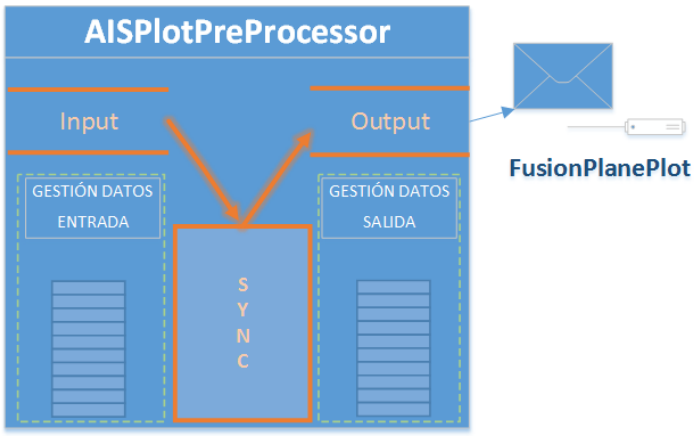
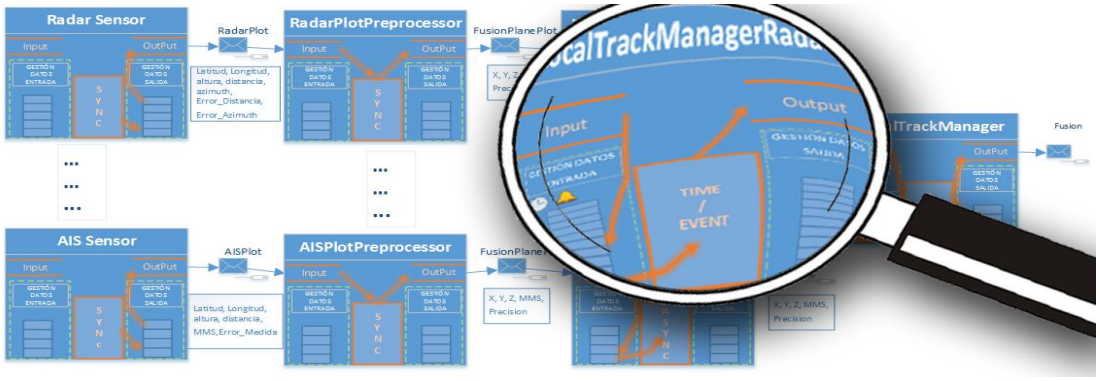


Ilustración 43. MessageProcessor AISPlorPreProcessor

En la Ilustración 43. MessageProcessor AISPlorPreProcessor, vemos que la configuración que se le ha establecido es la misma que la de RadarPlotPreProcessor, ya que nos encontramos en la misma situación en la que la transformación de las medidas, es un proceso que no conlleva mucho procesamiento y es poco probable encontrarse con algún bloqueo entre la recepción de las señales y su procesado.

5.3.3 Procesamiento local de Tracks



Una vez conseguido los mensajes de tipo **FusionPlanePlot,** donde se ha convertido las medidas a un eje de coordenadas, se pasa al procesamiento local, es decir, a la asociación los datos generados por cada uno de los sensores independientemente.

La función de asociación resolverá el problema de asignar a cada pista los nuevos datos adquiridos en el último intervalo de adquisición. Por tanto, el resultado de la asociación es un conjunto de datos asignados a las pistas centrales, y la selección de un conjunto de datos que no pertenecen a ninguna pista, potenciales fuentes de nuevas pistas.

El algoritmo de asociación debe resolver aquellos problemas en los que varios datos pueden pertenecer a varias pistas simultáneamente de manera que la asignación sea la mejor posible. Los problemas a resolver siempre son para los datos recibidos de un mismo sensor, por lo que en todo caso la asociación se resuelve por sensor, independientemente de la arquitectura de procesado. Esto implica que, aunque haya múltiples sensores actualizando la pista, el problema es “separable”, puesto que no hay ninguna dependencia entre datos de diferentes sensores.

El algoritmo de asociación se apoya en tres elementos: enventanado de datos y pistas, que permite descartar los pares incompatibles, asociación por código, que verifica cuando un dato puede directamente asignarse a una pista del mismo código, y asociación por distancia, que permite las asociaciones de mínima distancia. El último elemento busca el conjunto de asignaciones entre datos y pistas de mínima distancia, con el algoritmo de **Munkres**[2], mientras que los dos primeros elementos permiten reducir este último problema y aumentar la eficiencia, ya sea descartando a priori algunos pares mediante ventanas espacio-temporales, o produciendo asociaciones

[2] El algoritmo de Munkres resuelve la asignación de datos a pistas a través de la matriz de distancias.

directas por código cuando sea esto posible. Cuando las medidas de los sensores llevan asociadas un código de identificación, y la situación ofrece ciertas garantías, se utiliza para mejorar el proceso de asociación. Este proceso se lleva a cabo de forma independiente por cada uno de los sensores que proporcionan datos al sistema.

El último aspecto general importante de esta función de asociación es el referente a su temporización. La asignación es "secuencial", en el sentido de que cada dato se asocia en el intervalo en el que se recibe, como contraposición a otros algoritmos más complejos que realizan hipótesis con secuencias de más de un dato. Sin embargo, con objeto de evitar posibles errores por una asignación "precipitada", se impone un cierto tiempo de espera antes de asignar un dato, con objeto de asegurarnos de disponer de información suficiente. En cada intervalo de procesado se selecciona el conjunto de datos a asignar a las pistas existentes, que no coincide exactamente con el conjunto de datos recibidos a lo largo del periodo de adquisición transcurrido. Para mantener en cada momento el conjunto de datos a asignar que represente realmente todos los potenciales conflictos de asociación, se utilizan además aquellos datos del periodo de adquisición anterior que, aunque fueron asociados a una pista, no se asignaron finalmente y por lo tanto quedaron sin asociar. La razón de esto es evitar el tener que tomar una decisión en el instante de procesado acerca de un dato, sin tener aún toda la información necesaria para hacerlo de modo adecuado. Se considera un "tiempo de guarda", ΔT_{as}, que difiere la decisión de asociar un dato hasta tener la seguridad de que otros datos que pudieran llegar después no obligaran a tener que reconsiderar esta decisión.

Tras los resultados de las asociaciones explicadas anteriormente, resumiremos la función del MessageProcessor en las 3 funciones principales que puede realizar; la creación de una trayectoria, la actualización para asignar la detección a una trayectoria existente y el borrado de una trayectoria cuando desaparece de su alcance o no se detecta nuevas señales pasado un tiempo.

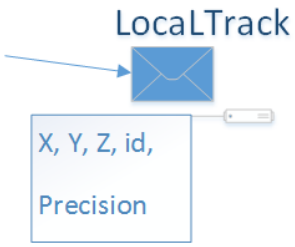


Ilustración 44. Mensaje LocalTrack

El mensaje resultante que enviaremos al siguiente MessageProcessor recibirá el nombre de **LocalTrack**, que como se ha detallado pertenecerá a un conjunto de detecciones de una misma trayectoria a la cual se le asigna un identificador de la trayectoria para conseguir relacionar la detección a una trayectoria.

Como es de suponer existen dos tipos de LocalTrackManager que por supuesto tienen como objetivo obtener los mismos resultados. Como en el resto de la cadena, estamos tratando datos que difieren leventemente en su contenido y el funcionamiento de ambos sensores es distinto, nos lleva a especializar dos tipos de MessageProcessor para adaptar de una manera más eficiente el procesamiento con respecto a los datos.

Antes de especificar los dos tipos de LocalTrackManager, me gustaría reflejar la gran diferencia en el procesamiento de los datos con respecto a las entidades que preceden en la cadena del sistema, es en este punto donde mayor procesamiento hay que realizar y donde se encuentra la mayor parte del problema real al que nos enfrentamos.

5.3.3.1 LocalTrackManagerRadar

En primer lugar mencionaremos el MessageProcessor LocalTrackManagerRadar, encargado de procesar los mensajes FusionPlanePlot procedentes de sensores de tipo radar.

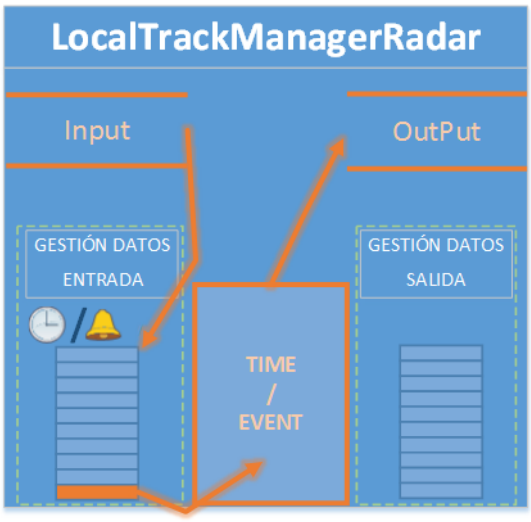


Ilustración 45. MessageProcessor LocalTrackManagerRadar

A modo de excepción, en la Ilustración 45. MessageProcessor LocalTrackManagerRadar, podemos ver como se ha representado dos tipos de gestión en los datos de entrada, lo que se quiere reflejar es la posibilidad de realizar la

gestión de los datos de entrada de ambas formar, por tiempo o por evento. Esto se debe al modo de funcionamiento de un radar.

En el caso de los radares tenemos dos posibilidades para comprobar cuando ha realizado una vuelta completa de todo su ciclo. En primer lugar, existe la posibilidad de trabajar por eventos gracias a que el radar cada vez que realiza un paso por norte emite un evento mediante el cual podemos detectar el momento exacto en el que el radar ha realizado una vuelta completa. Y en segundo lugar está la posibilidad de especificar un tiempo de espera para realizar el procesamiento ya que se conoce el tiempo exacto que tarda en realizar el ciclo completo.

Una vez realizado el procesado de los datos y llegado el momento de emitir los resultados, se puede ver que esto se hará de forma síncrona. Siguiendo la lógica en las demás entidades, se podría pensar que esta configuración es incorrecta y que convendría configurar una gestión de los datos de forma asíncrona, evitando de esta manera posibles bloqueos por las siguiente entidades, pero es aquí, donde se ve uno de los fuertes de este framework, donde al tener una configuración por tiempo en la siguiente entidad, la gestión de los datos se realizará en otro hilo de ejecución y será por este motivo por el que no inferirá en el procesamiento de la esta entidad.

5.3.3.2 LocalTrackManagerAIS

LocalTrackManagerAIS será la entidad encargada de asociar las nuevas detecciones a las trayectorias existentes o de crear una nueva en el caso de no existir. En el caso de los datos que proporciona los buques a las estaciones AIS nos encontramos con que cada uno de ellos facilita un identificador (MMS) con cada señal, mediante la cual resulta más simple la relación entre las diferentes señales recibidas. El estar identificadas no exime de realizar los cálculos correspondientes para comprobar su viabilidad y asociación.

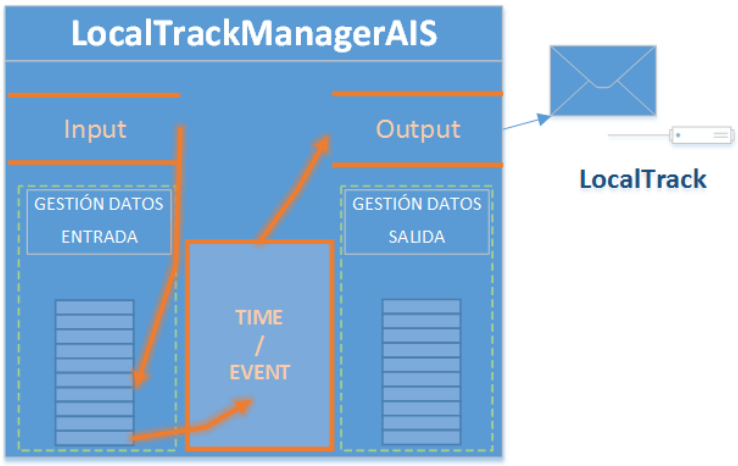


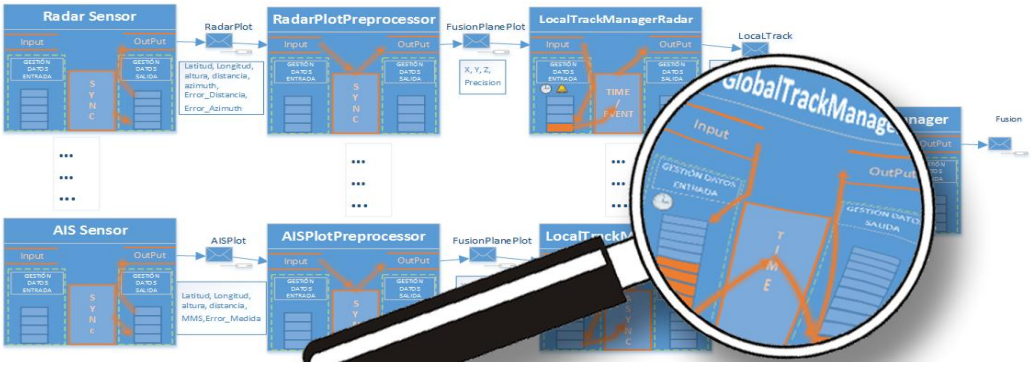
Ilustración 46. MessageProcessor LocalTrackManagerAIS

La Ilustración 46. MessageProcessor LocalTrackManagerAIS, nos representa la configuración que se le ha establecido a este tipo de MessageProcessor. Como se ha mencionado, este MessageProcessor recibirá los mensajes emitidos por AISPlotPreProcessor que reciben el nombre de FusionPlanePlot.

En la fase de la recepción de los datos podemos ver como se le ha especificado un procesamiento asíncrono para ir procesando continuamente las detecciones que reciba, pues este tipo de sensor no tiene ni ciclos, ni pasos por norte, ni realizar ningún tipo de barrido, este sensor recibe directamente las señales emitidas por cada buque.

Como en el caso de LocalTrackManageRadar la gestión de los datos de salida está configurada como síncrona por el mismo motivo.

5.3.4 Procesamiento global de Tracks



Llegados a este punto nos encontramos con que por cada sensor ya se ha convertido la señal recibida, ya se ha realizado la asociación de las detecciones por cada radar y es en este MessageProcessor donde se realizará la fusión general de todos los sensores, en el que se tendrán las trayectorias de todos los sensores y se relacionarán las trayectorias que hayan sido detectadas por más de un sensor.

Por lo tanto, como resultado final del procesado en la fusión de datos sale la estimación de las trayectorias de los objetos en el escenario. Este problema consiste en la estimación del movimiento de los objetos situados en la superficie marítima, y debe modelar de manera apropiada los sensores, considerando la naturaleza complementaria en la fusión, así como la peculiar maniobrabilidad de los blancos de interés (buques y pequeñas embarcaciones).

Según la arquitectura aplicada pueden plantearse diferentes soluciones para llegar a la estimación final de la trayectoria, según se realice una fusión de pistas o una fusión de plots. Como se ha mencionado en este caso se estiman primero las trayectorias con la información procedente de cada fuente y después se asocian y combinan

El modelo de los objetos caracteriza su maniobrabilidad, que podemos definir como la incertidumbre en la predicción, normalmente caracterizada como el intervalo de valores que puede tomar la magnitud de las posibles maniobras efectuadas por los objetos, impredecibles por el sistema de fusión. Habitualmente se modela como un proceso estocástico aceleración (con desviación típica σa), y puede orientarse a lo largo de la dirección del blanco, si consideramos más probable la existencia de maniobras longitudinales o transversales. En caso de no dar más probabilidad a ninguna de las dos, la incertidumbre es homogénea y viene dada por un círculo. Un sistema avanzado de filtrado puede además considerar como característica de interés la reducción de grados de libertad en la dinámica, restringida a las restricciones de

superficie y condiciones de maniobrabilidad en diferentes regiones de la superficie marítima.

Con un modelo de error aleatorio en la aceleración, la región de incertidumbre tiene un tamaño cuyos ejes crecen con el cuadrado del tiempo de predicción (y el área asociada crece como la cuarta potencia). Por esta razón el modelo de incertidumbre tiene bastante impacto en el alisamiento que realiza el filtro, y el parámetro σa se ha ajustado considerando el comportamiento con los datos y trayectorias disponibles.

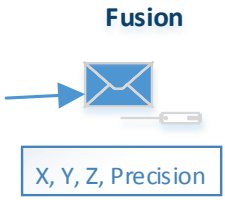


Ilustración 47. Mensaje Fusion

Como podemos ver representado en la Ilustración 47. Mensaje Fusion, al MessageProcessor que realiza esta acción se le asignará el nombre de **GlobalTrackManager** y el mensaje de salida que genere será **Fusion**.

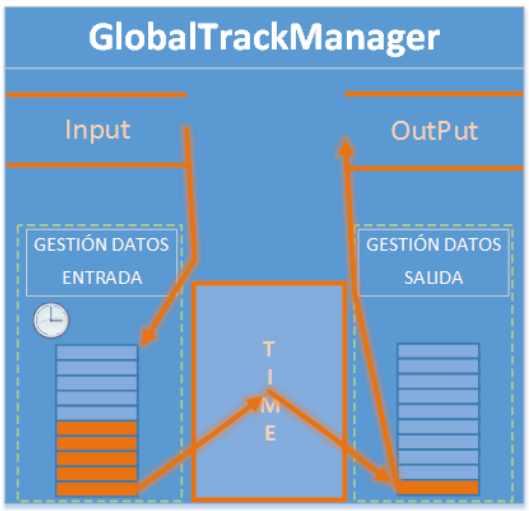


Ilustración 48. MessageProcessor GlobalTrackManager

La Ilustración 48, muestra como el MessageProcessor GlobalTrackManager por el lado de la gestión de los datos tiene un procesado por tiempo como se ha anticipado en el punto anterior y por la parte de la gestión de los datos de salida tendrá fijado una configuración asíncrona.

5.3.5 Visualización de información

Para la representación de los datos que procesa la aplicación, se ha decidido utilizar las utilidades que proporciona el Framework con el fin de poder representar las diferentes partes del sistema.

Existirán 3 utilidades para llevar a la práctica esta representación de los datos.

5.3.5.1 Player Link

Esta utilidad, se utiliza para establecer una comunicación a través de sockets con un programa externo al sistema, que se puede encontrar en cualquier otro dispositivo, con la única necesidad de contar con una conexión de red. Esta utilidad, se encargará de transmitir los datos al programa con un protocolo específico, permitiendo la comunicación para la representación los datos recibidos y procesados por el Framework.

Como los demás elementos del sistema, esta utilidad será un MessageProcessor más, con la peculiaridad de transmitir los datos mediante sockets.

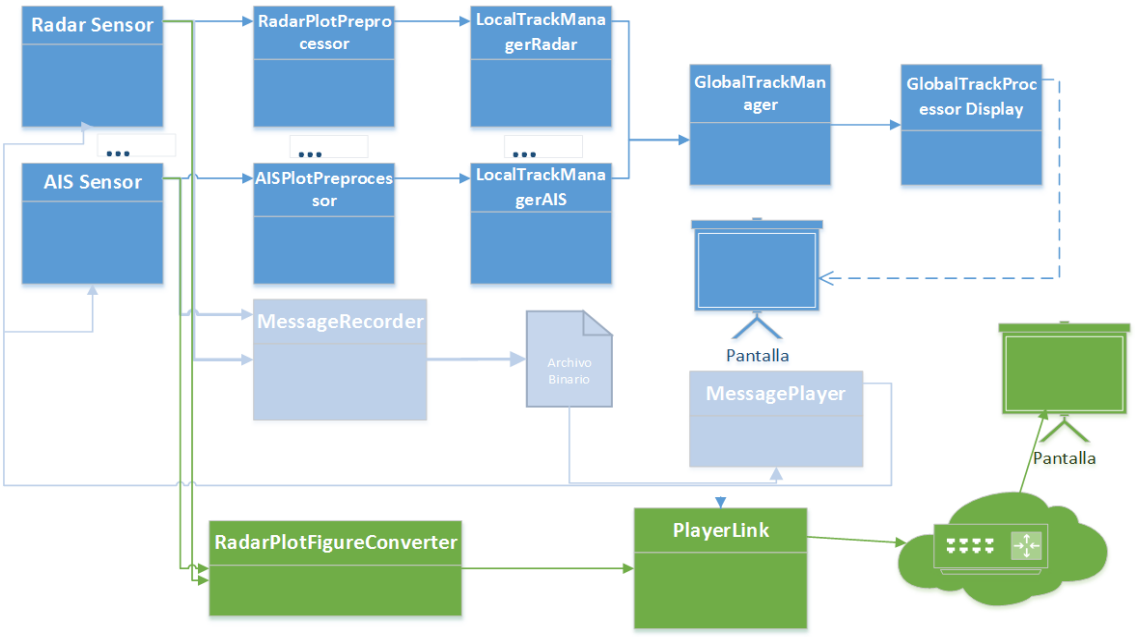


Ilustración 49. Utilidad PLayer Link

Como se puede apreciar en la Ilustración 49, Player Link está compuesto por un elemento intermedio más que recibe el nombre de **PlotFigureConverter**. El propósito de PlotFigureConverter, es el de transformar las señales recibidas en figuras geométricas como elipses, círculos, líneas de bearing, que tienen como objetivo, poder

representar por pantalla la información con la que se cuenta. A través de estas figuras, se podrá representar la posición en la que se encuentra los blancos, el margen de error de la medida y la trayectoria que el blanco ha seguido.

Otra aspecto que se puede apreciar en la Ilustración 49, es donde se realiza la conexión de esta utilidad, en este caso se puede ver que los resultados por pantalla serán exactamente las detecciones realizadas por los sensores, detecciones que todavía no han sido tratadas por el framework. La utilidad Player Link nos da la posibilidad de representar ambas señales, las recibidas por los sensores y las señales tras la fusión de los datos, con lo que podremos comparar ambos resultados.

5.3.5.2 Entrada

Para conseguir reproducir los datos que el Framework recibe, se ha añadido a la salida de todos los RadarSensor y AISSensor un MessageProcessor llamado **MessageRecorder,** que se encargará de serializar cada uno de los mensajes recibidos por los sensores en un fichero binario. Además de la información proporcionada por los sensores, se almacena una serie de campos como el identificador del sensor que ha detectado esta señal o el tiempo en el que ha detectado esta señal, para a posteriori facilitar la representación de estos datos.

Por lo tanto, el resultado del sistema tras añadir esta utilidad será la siguiente.

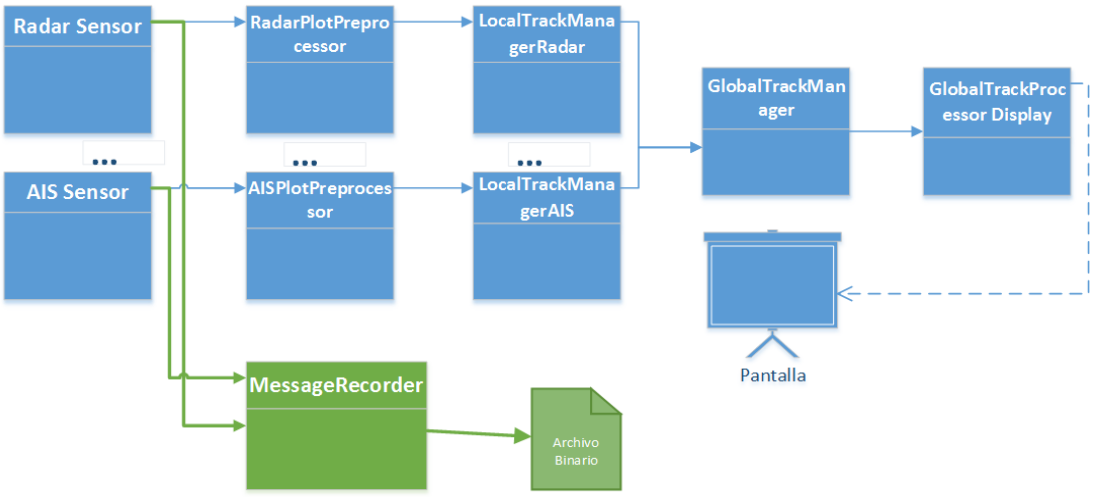


Ilustración 50. Utilidad Recorder

En la Ilustración 50, se puede ver una vez más la posibilidad de interconexión del framework, como los datos generados por los sensores puede recibir diferentes procedimientos simultáneamente con la interconexión de las utilidades y el propio sistema.

5.3.5.3 Salidas

Esta utilidad, sirve para interpretar los datos que han sido almacenados previamente con la utilidad MessageRecorder y permitir volver a representarlos.

Esta utilidad, permite representar la situación del Framework de igual manera que si se estuviesen recibiendo los datos desde un sensor real, ya que se han almacenado todos y cada una de las detecciones generadas por los sensores.

Al igual que la utilidad Recorder, esta utilidad es un MessageProcessor y recibe el nombre de **MessagePlayer**.

Para el funcionamiento de esta utilidad, en primer lugar, se leerá de fichero una entrada y se detectará de que sensor ha sido generada esa señal, tras su identificación en vez de enviar un mensaje al MessageProcessor AISSensor o RadarSensor para que este envíe la señal, se realizará el envío directamente desde el sensor que genero la señal, de esta manera, conseguimos que los datos estén dentro del sistema y se procesen para generar los resultados de nuevo.

Al utilizar la utilidad MessagePlayer, el sistema quedaría de la siguiente manera.

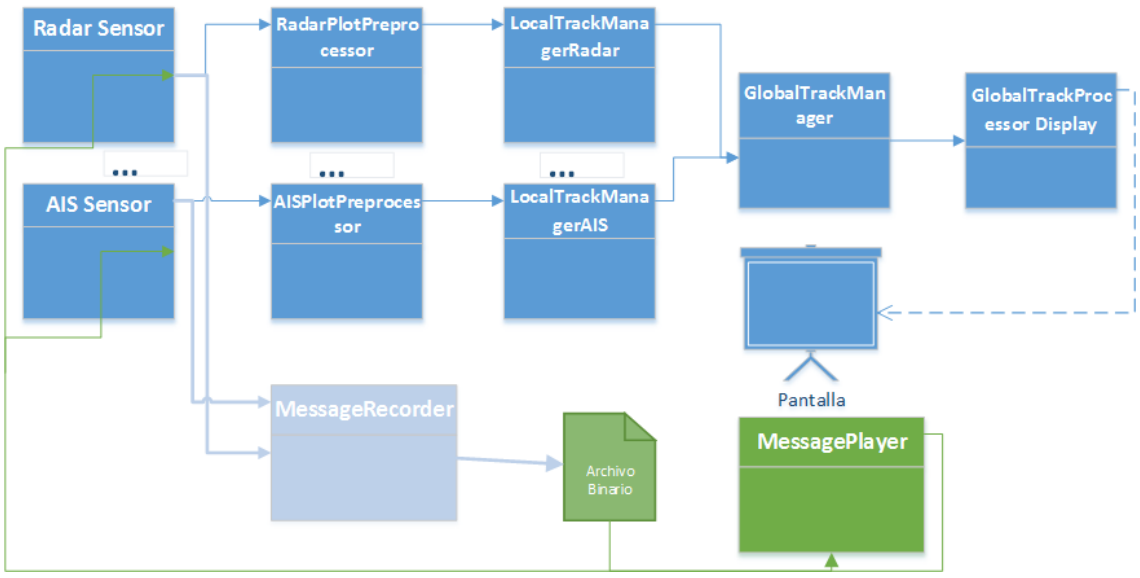


Ilustración 51. Utilidad MessagePlayer

Se puede observar como la utilidad lee de fichero en la fase de procesamiento los datos y se encarga de enviar a través del sensor correspondiente.

5.3.5.4 Data player

Por último, tenemos una utilidad de mayor complejidad que se encarga de representar los datos en una herramienta en formato gráfico, donde se podrá ver en tiempo real lo que está ocurriendo en el escenario del problema.

El programa Data Player, será al cual el Player link se conecte y envíe los datos. Por lo que tras haber explicado y conocer el funcionamiento de Player Link, sabremos que este programa recibirá a través de una comunicación por sockets y mediante un protocolo determinado, figuras geométricas que corresponderán con las detecciones y posiciones de los blancos.

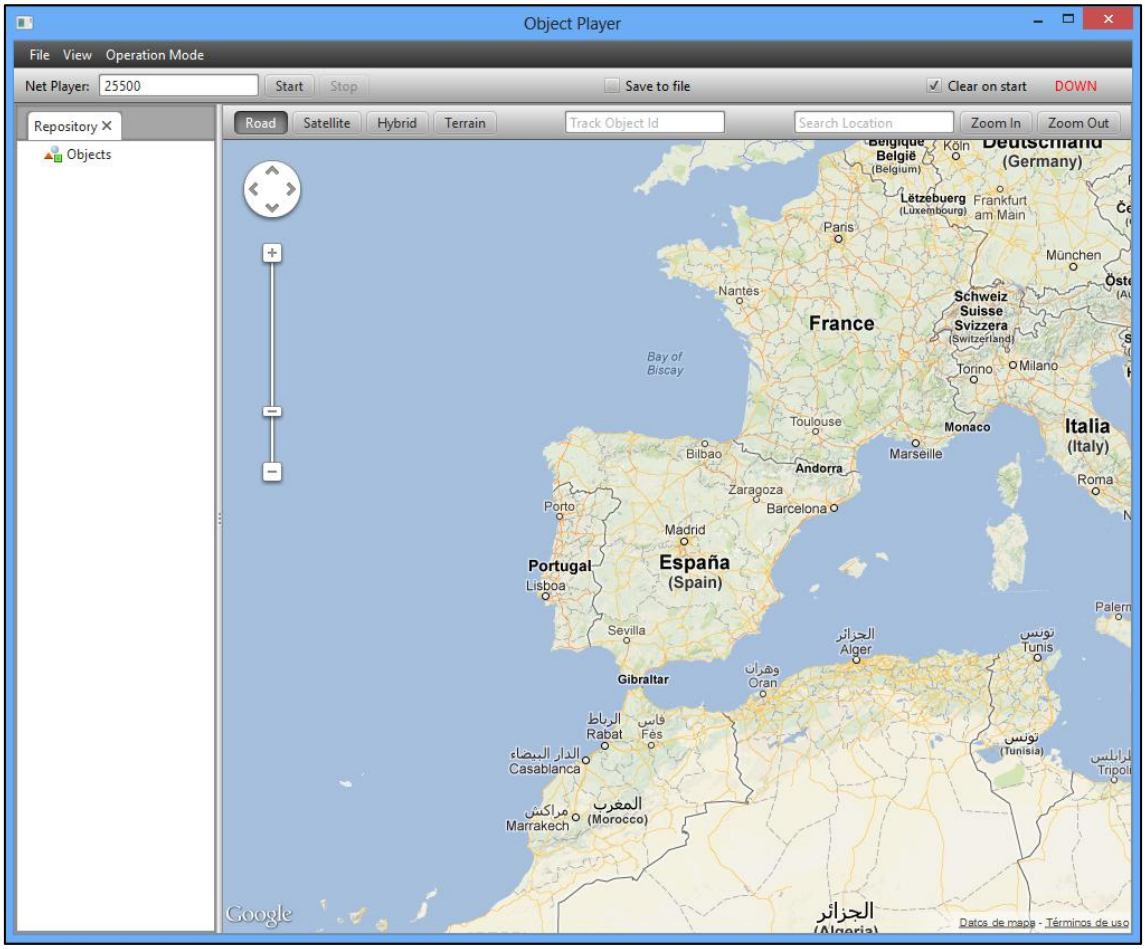


Ilustración 52. Interfaz Data Player

En la Ilustración 52, podemos ver cómo será la interfaz de esta herramienta, la cual aparte de proporcionar la representación los datos recibidos mediante una conexión, dispone de muchas otras posibilidades. Alguna de las posibilidades que tiene la aplicación, es la creación de blancos y trayectorias para conseguir escenarios personalizados, la lectura directa de un fichero que contenga las detecciones, y por

supuesto la opción de la cual haremos uso en este sistema, que será la representación de la información recibida por una comunicación por sockets.

A continuación, a modo de ejemplo para proporcionar una leve noción de la información que se facilita al programa, se adjunta un fragmento del protocolo para la representación de una señal recibida por un sensor AIS.

```
ts:270000;type:object;object_id:A013;category:AIS;object_operation:set
;object_type:circle;circle_lat:-33.018166;circle_lon:-
71.586832;circle_radius:100;object_tag:A013;object_color:#FF0000;\r\n
```

Tabla 16. Protocolo Representación Player Link

En la Tabla 16, podemos destacar algunos campos como el objeto circle, el cual tendrá los campos latitud, longitud que corresponderán con la posición en el mapa y un radio para su tamaño. Con este círculo se representará la posición del blanco y con el tamaño el margen de error que tiene la detección en cuestión.

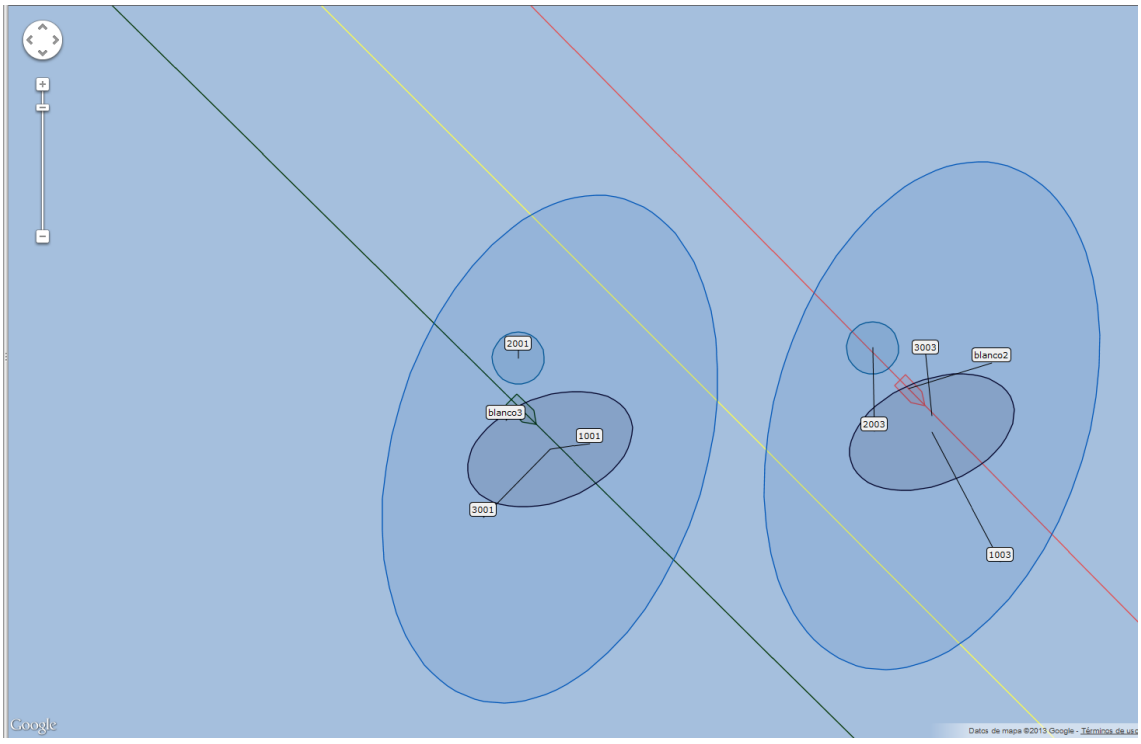


Ilustración 53. Ejemplo Data Player

En la Ilustración 53, podemos ver que el escenario consta de dos blancos y que estos blancos están siendo detectados por 3 sensores, en concreto podemos decir que son dos sensores radar y un AIS. Esto se puede saber porque los sensores AIS representan su detección mediante un círculo y porque a no ser que nos encontremos

con una situación fuera de lo normal, la precisión de las medidas AIS tienen menor error, lo que supone un tamaño más pequeño de la figura.

En el escenario, se podrá incluir la configuración de los sensores que se encuentran en la zona y poder ver así, los puntos de intersección entre los diferentes radares su ciclo de giro y las detecciones que genera cada uno.

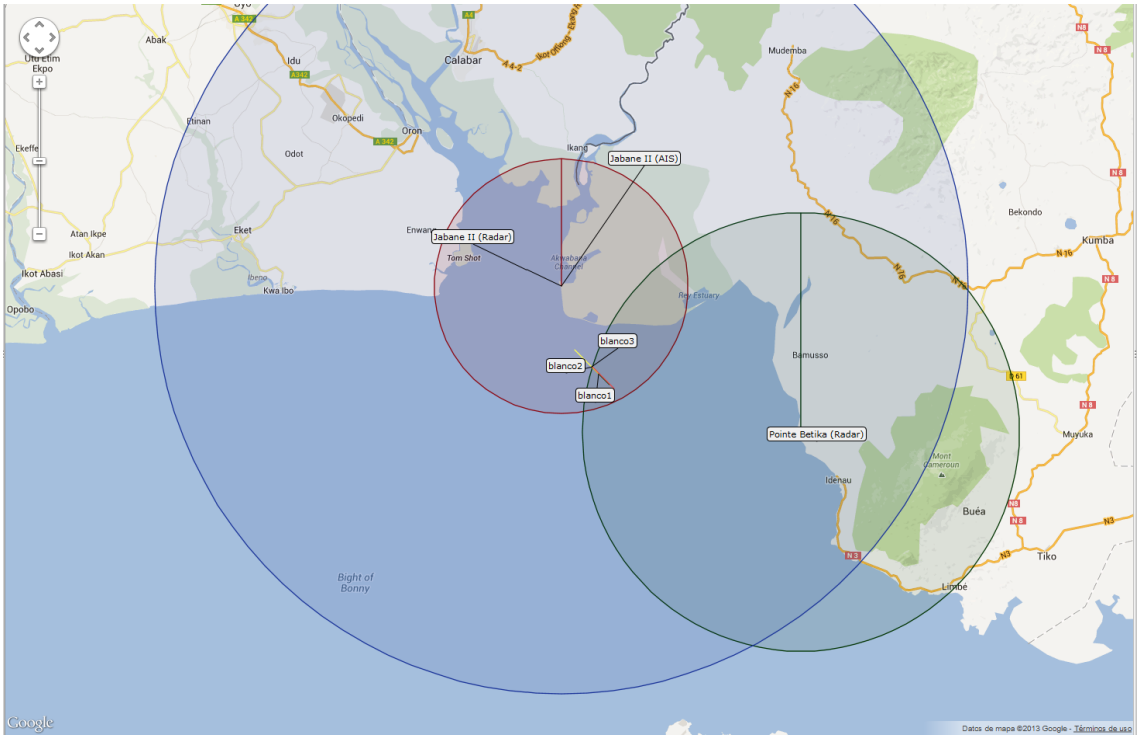


Ilustración 54. Ejemplo Escenario Data Player

Podemos ver en la Ilustración 54, como esta zona en cuestión está formada por 3 sensores, el ámbito de cada uno de ellos y las zonas en las que se solapan los sensores.

5.4 Pruebas de Implementación del Sistema

Tras la configuración del todo el sistema se llevará a cabo una serie de pruebas que permitan ver y demostrar el funcionamiento del mismo.

Para llevar a cabo estas pruebas, se hará uso de la aplicación del framework en el entorno de la fusión de los datos.

En él, nos encontraremos con las entidades detalladas en la sección 5.3 Diseño de los componentes que realizarán las funciones especificadas sus puntos.

Para la representación de los datos se hará uso de las utilidades Player Link que se conectará con el Data player ubicado en otra máquina.

5.4.1 Prueba 1

En esta prueba se podrá ver como al conectar el Player Link sobre la salida de los mensajes generados por los MessageProcessor FusionPlanePlot, se puede representar en la aplicación Data Player, las detecciones de los diferentes sensores con los que cuenta el escenario.

Para facilitar la comprensión de la prueba se adjuntará un diagrama del sistema completo con la interconexión de las diferentes utilidades.

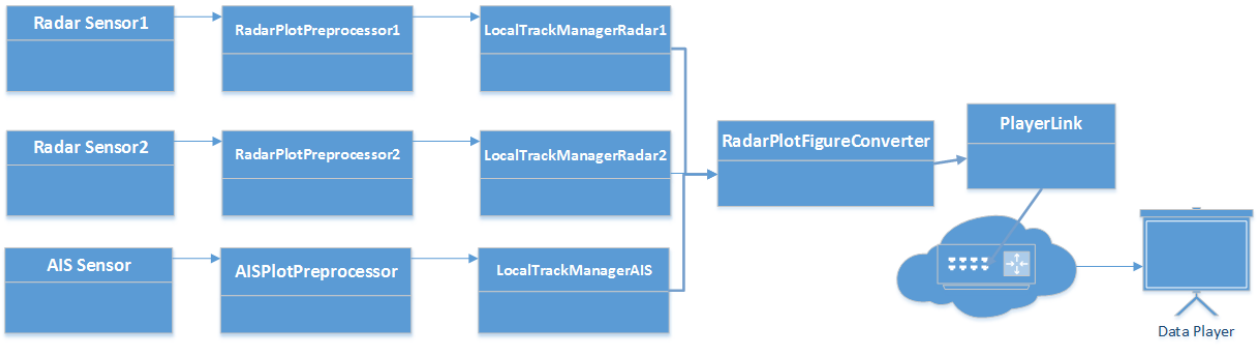


Ilustración 55. Diagrama Prueba 1

Podemos ver a través de la Ilustración 55, como los datos son recibidos por dos radares y un sensor AIS que tras su procesado local, realiza una conversión a figuras geométricas con el MessageProcessor PlotFigureConvertes, los cuales serán enviados con la utilidad PlayerLink al Data Player donde veremos los resultados.

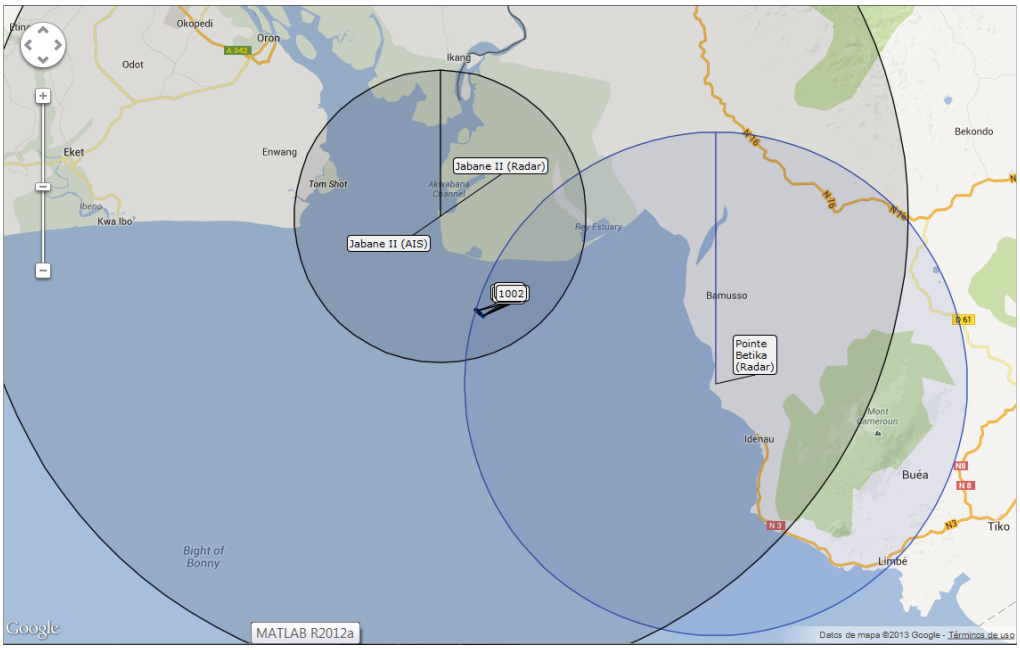


Ilustración 56. Escenario Prueba 1

La Ilustración 56 representa el escenario que vamos analizar, en él podemos ver los tres sensores definidos en el sistema detallado anteriormente y la cobertura de cada uno de ellos.

En el medio de la imagen se puede apreciar una pequeña mancha donde apunta el identificador 1002, esto corresponde con una detección de un sensor, por lo tanto vemos que será donde se encuentren los blancos del escenario, justo en la intersección de los 3 sensores.

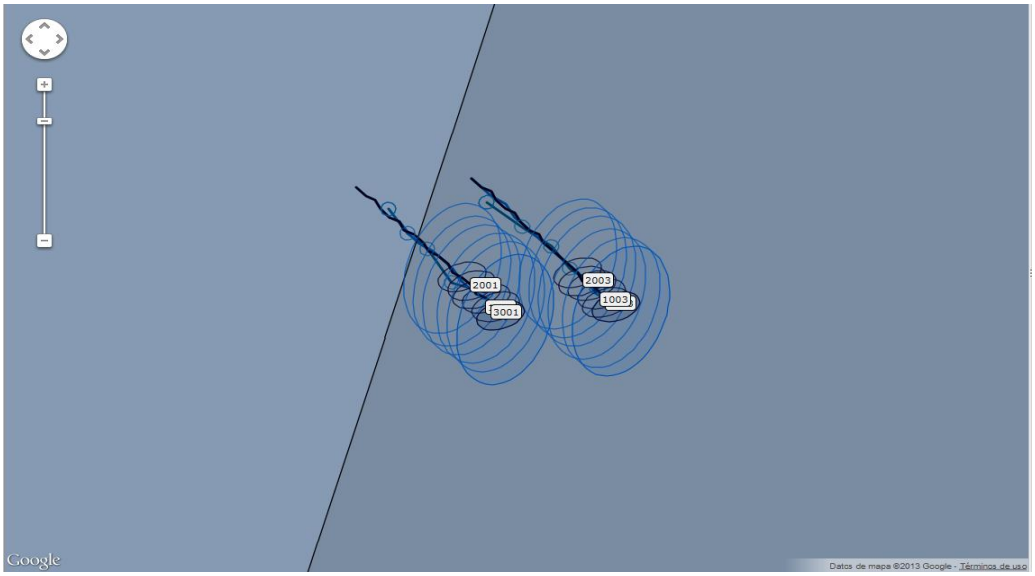


Ilustración 57. Salida Prueba 1

Si analizamos la Ilustración 57, corresponde con la sección central de la imagen anterior, donde se mostraba el escenario al completo. En ella, podemos ver como la trayectoria de la izquierda en un principio no se encuentra en la cobertura de uno de los sensores y es por ello que solo aparecen detecciones de dos de los sensores hasta el momento en el que entra en la parte de cobertura del 3º sensor.

Otra característica importante a destacar en la imagen, es la diferencia entre los tamaños de las elipses. Esto se debe por la distancia a la que se encuentran los blancos del sensor, que como es de suponer, a mayor distancia mayor será la posibilidad de error en la precisión de la detección. Si combinamos las dos imágenes anteriores podemos ver como el color de las detecciones corresponden con el color del sensor. Por lo tanto, los elipses de color negro corresponden con las detecciones del Radar Janabe II, que al estar más cerca, son menores que los elipses azules, que corresponden con el Radar Pointe Betika, las cuales, se están a una distancia mucho mayor.

Tras la ejecución de la prueba 1, podemos ver como los resultados obtenidos han sido capaces ser generados en tiempo real, sin pérdida de información y como tras la implementación del problema con el framework, se cumplen los objetivos del proyecto.

5.4.2 Prueba 2

En esta prueba, se pretende realizar una demostración más completa, donde se contemple todos los aspectos comentados a lo largo del framework, destacando las posibilidades de interconexión de nuevos elementos en el sistema sin repercutir en el funcionamiento normal del sistema.

Partiendo de la base del ejemplo anterior, se puede ver en la Ilustración 58. Diagrama Prueba 2 , como tras el procesado de RadarPlotPreProcessor, se ha incluido un nuevo MessageProcessor. Este MessageProcessor recibirá el nombre de **MessageFilterZone**.

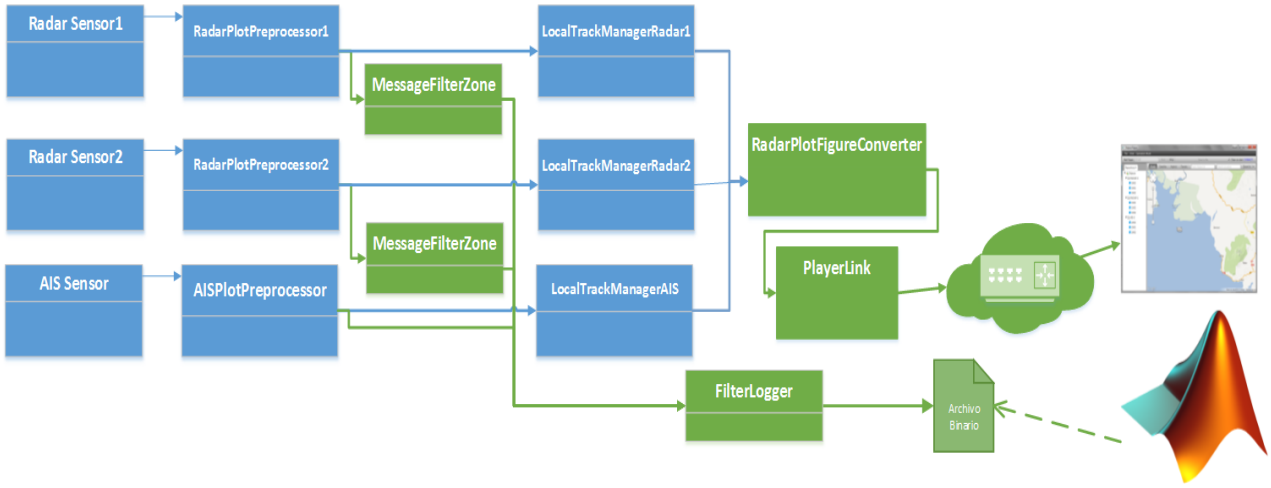


Ilustración 58. Diagrama Prueba 2

Como se puede ver en la ilustración, el nuevo MessageProcessor MessageFilterZone estará conectado a la salida de preprocesado de los radares. En este momento, como se ha detallado en Preprocesado de la información, dispondremos de las detecciones de los blancos en medidas cartesianas.

Con esta información, MessageFilterZone se encargará de realizar un filtrado de las señales recibidas de los radares sobre una zona del escenario en concreto. Este tipo de filtrado, puede ser de interés en aquellas zonas en las que la cobertura de los radares tenga un alto nivel de interferencias, por lo que los datos que nos proporcionan, no son del todo fiables.

Por lo tanto, durante la zona filtrada, solo tendremos en cuenta a la hora de visualizar las detecciones, las señales recibidas por los sensores AIS, que en un principio, la información que proporcionan es más fiable.

Una vez filtrada la zona, necesitaremos hacer uso de un nuevo MessageProcessor al que proporcionando el resultado de los datos, se encargue de almacenar en un fichero las detecciones válidas. Este nuevo MessageProcessor se le ha asignado el nombre de FilterLogger, el cual, se encargará de almacenar los datos en fichero con un formato, que más tarde sea posible a través de la aplicación Matlab, la representación de los datos.

Para ver todas las partes de la prueba, se adjuntará a continuación partes del código representativas con el fin de conseguir entender, además de la configuración establecida, la manera de llevarlo a cabo.

```
AISPreProcessor->attachProcessor(FilterLogger);
radarPreProcessor->attachProcessor(MessageFilterZone);
MessageFilterZone ->attachProcessor(FilterLogger_);
```

Código 16. Conexión MessageProcessor Prueba 2

En primer lugar, en el Código 16, se puede ver como con 3 simples líneas de código se consigue establecer la conexión de las nuevas entidades sin crear ninguna modificación al resto de la configuración en el sistema. Si recordamos la función del método mencionado en el punto 4.4.1.1.1 AttachProcessor(MessageProcessor*), con él establecemos la conexión entre ambas entidades.

Por lo tanto, vemos en la primera línea de código que tras el pre-procesado de las señales AIS enviamos los mensajes directamente al MessageProcessor FilterLogger, donde se almacenará estos datos en el fichero que posteriormente utilicemos en Matlab para representar los datos.

En la segunda línea de código, estamos estableciendo la comunicación entre el pre-procesado de los sensores radar y el MessageFilterZone, el cual como veremos en la siguiente fragmento de código se encargará de filtrar las señales de una zona en concreto.

Por último en la tercera línea estableceremos la conexión entre MesageFilterZone con el MessageProcesso FilterLogger. Con esta conexión todos los mensajes que emita MessageFilterZone llegarán a FilterLogger que los almacenará en el fichero.

```
void MessageFilterZone::processIncomingMessage(shared_message message){
boost::shared_ptr<FusionPlanePlot> fusionPlanePlot =
boost::static_pointer_cast<FusionPlanePlot>(message);
	Point3D& punto = fusionPlanePlot->getPosition();
bool enviar = (punto.getX() < 7000 || punto.getX()>8000) && (punto.getY() < -17500 ||
punto.getY()>-16700);
	if(enviar){
		sendMessage(message);
	}}
```

Código 17 . Filtrado MessageFilterZone

En el Código 17, se puede ver que el filtrado realizado en MessageFilterZone es bastante simple. Para llevarlo a cabo, se han fijado una serie de coordenadas que corresponden con la zona de filtrado, en las cuales, si la señal recibida proviene del interior de esa zona, la variable **enviar** obtendrá el valor false y por consiguiente, tras la comprobación del if, no entrará en él y el mensaje no será enviado. Con esta simple acción, conseguiremos que en el fichero solo se almacene las señales que están fuera de la zona.

Una vez comprendido el problema, se mostrará los resultados obtenidos, donde podremos ver el correcto funcionamiento de la nueva implementación configurada.

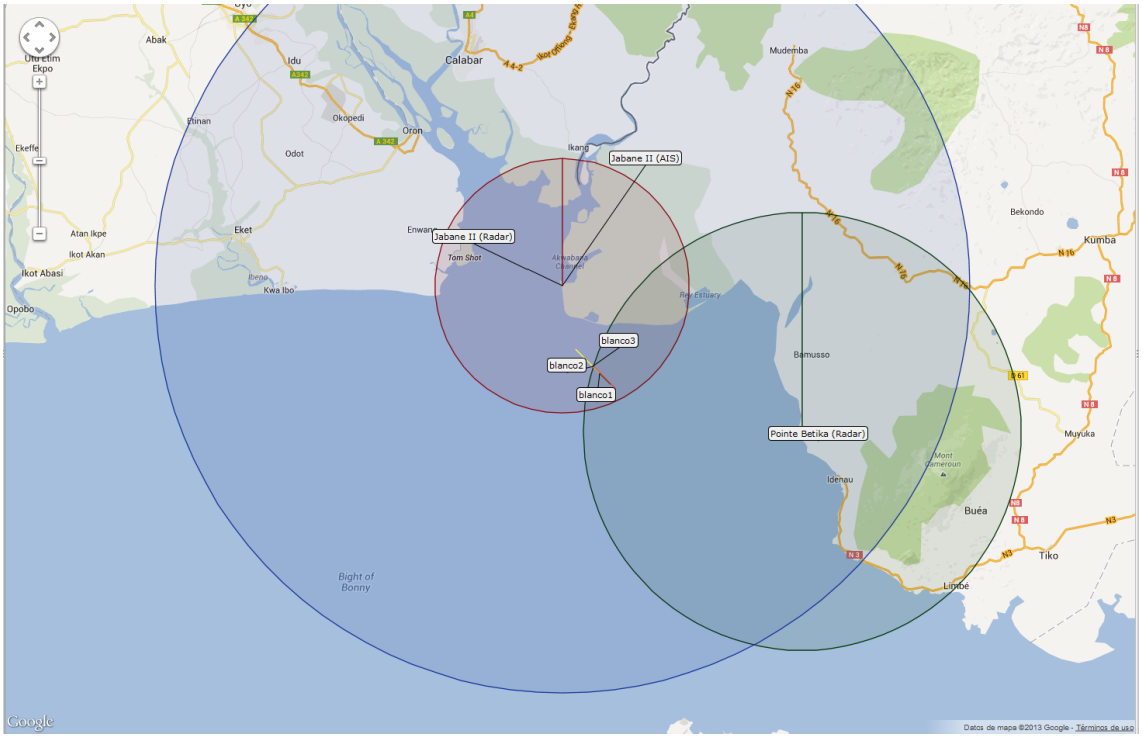


Ilustración 59. Escenario Prueba 2

Para ponernos en situación, la Ilustración 59 representa el escenario donde se realizará las detecciones.

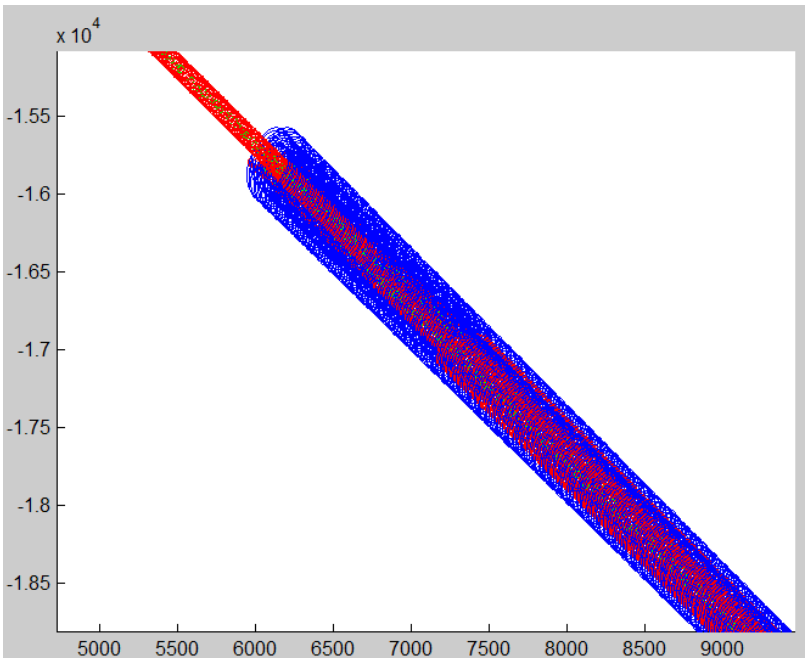


Ilustración 60. Representación Detecciones en Matlab sin Filtrado.

La Ilustración 60, muestra los resultados obtenidos en la aplicación Matlab tras la ejecución del escenario sin realizar el filtro zonal. Se puede ver las detecciones de todos los sensores a lo largo de todas las trayectorias capturadas. Esto nos servirá para una vez aplicado el filtro, comprobar la diferencia que habrá.

Con el fin de facilitar la representación que se ha llevado a cabo en la aplicación Matlab, se ha hecho un zoom sobre los resultados, donde se podrá visualizar con más detalle las diferentes figuras que hay.

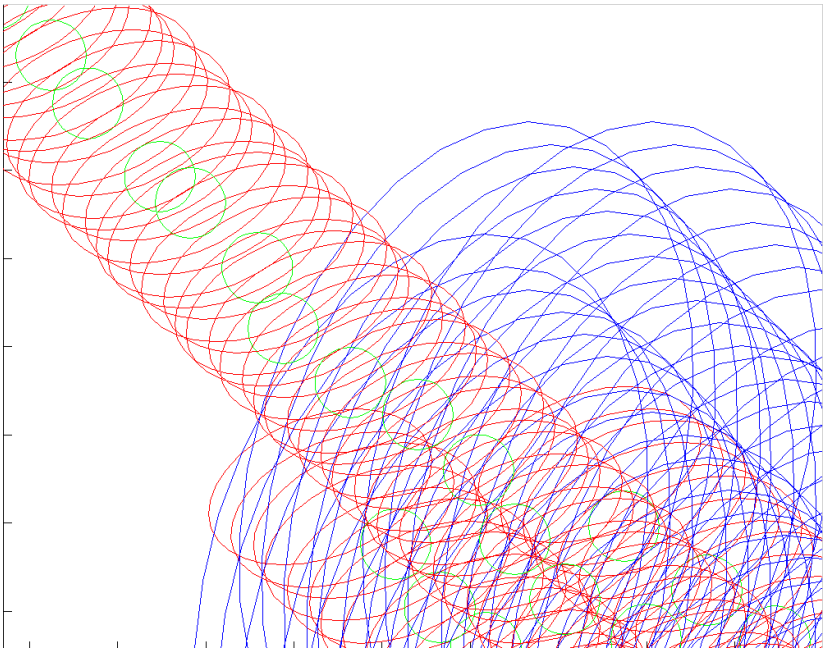


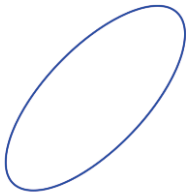
Ilustración 61. Zoom Representación Matlab

Se puede apreciar en la Ilustración 61, que se distingue tres tipos de figuras, dos elipses y un círculo. Cada una de estas figuras, corresponde con las detecciones que ha recibido en cada uno de los sensores.

Las imágenes por separado con las que nos encontramos son las siguientes.

AIS Janabe II: este sensor AIS representará sus señales detectadas a través del circulo verde que podemos ver en la imagen.

Radar Janabe II: este sensor Radar representará sus detecciones mediante una elipse roja como vemos en la imagen de la derecha.



Radar Pointe Betika: Este sensor de tipo Radar representará sus detecciones con la figura geométrica elipse de color azul.

Una vez entendido la representación adoptada en la aplicación Matlab, estaremos en disposición de ver los resultados obtenidos tras el filtrado que se le ha aplicado con la configuración establecida en el sistema.

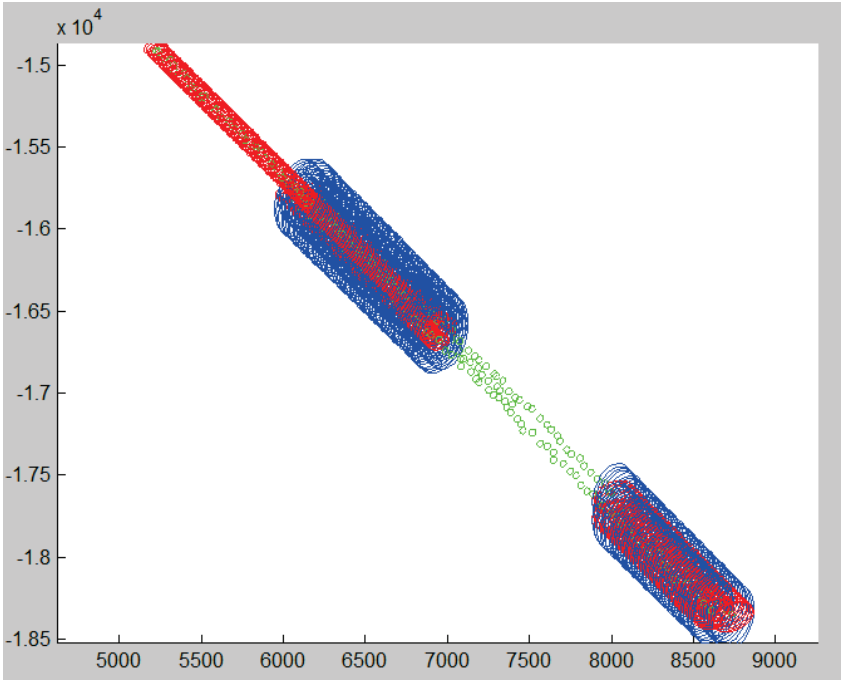


Ilustración 62. Representación Detecciones Matlab Tras filtrado

Viendo la Ilustración 62, se puede apreciar perfectamente como en la zona que se le ha aplicado el filtrado de los mensajes, no aparecen ninguna de las detecciones recibidas de los radares, quedando únicamente las detecciones del sensor AIS que se encuentra en el escenario.

Como conclusión final se puede ver como los resultados obtenidos, son totalmente satisfactorios. Creo personalmente, que gracias a esta prueba se ha podido reflejar muchas de las funcionalidades que puede proporcionarnos el framework, con una fácil configuración, y haciendo hincapié, en la rápida adaptación del nuevo sistema sobre uno ya implementado, todo ello, sin repercutir al funcionamiento del mismo.

Una vez más, el sistema ha conseguido responder en tiempo real, con un rendimiento óptimo, todo ello gracias a las posibilidades que ofrece el framework, con el reparto en la carga de trabajo de forma paralela, la gestión eficiente de los mensajes transmitidos y la modularidad que proporciona.

Conclusiones y Futuros Trabajo

6 Conclusiones y Futuros Trabajos

6.1 Conclusiones

Ante la necesidad de realizar un procesado intensivo de los datos, en un tiempo de respuesta real, se ha llevado a cabo la creación de un framework, que tras su desarrollo, se ha podido demostrar cómo se ha resuelto el problema de una forma totalmente satisfactoria.

A lo largo del documento, se ha podido ir comprobando cómo se han ido resolviendo cada uno de los objetivos, y cómo, gracias a la implementación del framework sobre la fusión de los datos, se ha demostrado la generalización que ofrece el framework para ser aplicado ante cualquier problema que incluya un procesado de los datos.

Además, se ha comprobado la posibilidad de proporcionar un paralelismo sobre el procesamiento de los datos a nivel de sensor, consiguiendo de esta manera, un rendimiento óptimo en el sistema.

Otros de los objetivos que se ha podido demostrar, es la posibilidad extender el sistema tanto como sea necesario, a través de la flexible interconexión que proporciona el MessageProcessor con el uso de signals y slots de la librería Boost.

También, cabe destacar la gran gestión de los datos que se ha proporcionado en el framework, gracias a la implementación en el MessageProcessor sobre la gestión en los datos de entrada y salida.

Aparte de cumplir con todos los objetivos planteados, se ha conseguido mediante la utilización de la tecnología seleccionada, un framework multiplataforma, el cual permite su uso, en diferentes sistemas operativos.

Por tanto, los principales beneficios que se han conseguido con la creación de este proyecto y con la implementación del framework, son la posibilidad delegar; la gestión de las comunicaciones entre las entidades, la gestión del paralelismo en el sistema y la gestión de la memoria, en las funcionalidades que ofrece el framework. Consiguiendo dotar al sistema que lo implemente de una forma eficiente, todas estas necesidades que requieren la mayoría de los sistemas de hoy en día, sin necesidad contemplarlas como parte del problema, pudiendo centrar el desarrollo, en los objetivos y en el procesamiento que requieran los datos.

6.2 Trabajos Futuros

Al haber generado un framework tan genérico, que permita adaptarse a cualquier sistema donde su principal objetivo sea el procesado de la información, surge la posibilidad aprovechar el proyecto como base para añadir nuevas funcionalidades y ampliarlo, o adaptarlo para aplicarlo a entornos con otro tipo de objetivos.

Alguna de las ideas que se propone con el fin de ampliar este proyecto, es la de incluir al framework una utilidad que permita representar la carga de trabajo y el estado de las colas de gestión de los datos de entrada y salida de cada MessageProcessor. Esto, permitirá al sistema que se implemente, establecer una mejor configuración sobre en los MessageProcessor, gracias a poder visualizar en tiempo real, donde se está produciendo un mayor procesamiento y las posibles congestiones en las colas de gestión de los datos de entrada y salida de cada MessageProcessor.

Otra línea de trabajo que se plantea, es la creación de un manual de uso del framework, con el fin de proporcionar las posibilidades de configuración que ofrece el framework y un detalle de cada una de las implementaciones necesarias para configurar un sistema con el framework.

7 Presupuesto

El coste del análisis, diseño y desarrollo del proyecto viene establecido por la duración del mismo y por otros gastos indirectos. Todos estos gastos se desglosarán a continuación, teniendo en cuenta los siguientes datos:

Datos de personal, 2013				
Duración del proyecto (días)	Horas diarias	Dedicación hombre mes	Coste hombre mes	Coste (Euro) [1]
98	4	131,25	2.391,26	7.141,90
			Total	***7.141,90***

[1] Fórmula del cálculo del coste de personal
Coste= ((duración días * horas días) / dedicación hombre mes) * coste hombre mes

Tabla 17. Presupuesto-Datos de personal, 2013

Fuente: INE, Encuesta Trimestral de Coste Laboral (INE, 2013)

Además, para la realización del proyecto ha sido necesario adquirir los siguientes equipos con su correspondiente coste de amortización:

Equipos					
Descripción	Coste(Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de deprecación	Coste imputable [2]
Ordenador Intel Core i7	480,00	100	5	60	40,00
				Total	***40,00***

[2] Fórmula de cálculo de la Amortización:

$\frac{A}{B} \times C \times D$ **: (Vicinv)**

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

Tabla 18. Presupuesto-Equipos

Por otro lado se van a detallar otros gastos directos del proyecto que no han sido contemplados en los conceptos anteriores como pueden ser fungibles, viajes y dietas, otros…

Otros gastos directos		
Descripción	**Empresa**	**Costes imputables**
Internet	Telefónica	200,00
Luz	Iberdrola	100,00
Microsoft Office 2010 Estudiantes	Microsoft	140,00
Microsoft Visio Professional 2013	Microsoft	739,00
	Total	***1.179,00***

Tabla 19. Presupuesto-Otros gastos directos

Además, no se ha necesitado ninguna subcontratación para realizar alguna de las tareas del proyecto y hemos tenido en cuenta una tasa del 20% para calcular los gastos indirectos.

El coste total del proyecto, de forma resumida, lo podemos comprobar en la siguiente tabla:

Resumen de costes	
Descripción	**Presupuesto costes totales**
Gastos de personal	7.141,90
Gastos de equipo (Amortización)	40,00
Subcontratación de tareas	0
Otros gastos directos	1.179,00
Gastos indirectos (20 %)	1.672.18
Total Sin IVA	**10.033,08**
Total Con IVA (21 %)	**12.141,03**

Tabla 20. Presupuesto-Resumen de costes

El presupuesto total del proyecto asciende a la cuantía de **DOCE MIL CIENTO CUARENTAY UN EUROS CON TRES CÉNTIMOS DE EURO**.

8 Bibliografía

ansi. (02 de 12 de 2005). Recuperado el 2013, de http://publicaa.ansi.org/sites/apdl/Documents/Standards%20Action/2005%20PDFs/SAV3648.pdf

Austern, M. (24 de 06 de 2005). *Open-std* . Obtenido de http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1836.pdf

Boost. (04 de 02 de 2013). Obtenido de http://www.boost.org/users/history/version_1_53_0.html

CICESE. (2002 йил 02). *Fundamentos de Paralelización*. From http://telematica.cicese.mx/computo/super/cicese2000/paralelo/Part2.html

Elmue. (s.f.). *codeproject*. Obtenido de http://www.codeproject.com/Articles/6780/Type-safe-Signals-and-Slots-in-C-Part-2

García , J., Gueerrero, J. L., A. L., & Molina, J. M. (s.f.). *Universidad Carlos III*. Obtenido de http://www.uc3m.es/portal/page/portal/actualidad_cientifica/noticias/vigilancia_maritima

Gregor, D. (04 de 05 de 2004). *Boost 1.52.0*. Obtenido de http://www.boost.org/doc/libs/1_52_0/doc/html/signals.html

INE. (2013). *Instituto Nacional de Estadística*. Recuperado el 6 de 2013, de http://www.ine.es/jaxiBD/tabla.do

ISO. (1 de 09 de 2011). Obtenido de http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=50372

ISO/IEC. (01 de 09 de 2011). *IEC*. Obtenido de http://webstore.iec.ch/webstore/webstore.nsf/Artnum_PK/45491

Java Data Processing Framework. (s.f.). Obtenido de http://www.jdpf.org/

Java Media Framework. (s.f.). Obtenido de http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html

Moore, G. E. (1965). *IEEE Global History Network*. Obtenido de http://www.ieeeghn.org/wiki/index.php/Gordon_E._Moore

OSGI. (s.f.). Obtenido de http://www.osgi.org/Main/HomePage

Vicinv. (s.f.). Obtenido de http://vicinv.ujaen.es/files_vicinv/GU%C3%8DA%20PARA%20EL%20C%C3%81LCULO%20DEL%20PRECIO%20DE%20LOS%20CONTRATOS%20(v3).pdf

Wikipedia. (s.f.). Obtenido de http://es.wikipedia.org/wiki/Puntero_inteligente

Williams, A., & Botet Escriba, V. J. (12 de 2011). *Boost*. Obtenido de http://www.boost.org/doc/libs/1_52_0/doc/html/thread.html

Fin de Documento.